6 Models for Learning with an Hierarchical Structure

1 A Family of Models for Code Learning Experiments

The family of models described in this chapter has an hierarchical structure built into it as a convenience both for observation and in wedding the model to the experimental or tutorial situation it represents. If one recalls the caveats of the last chapter, this structure is (as it is said to be) a convenience. All the same, comparable structures exist in the design of adaptive teaching and training machines.

The model is also partitioned into a component called the learning model and a component called the teaching model. The latter component is trivial; it simulates the mechanism described in Chapter 5 and is an elaboration of the steady state regulators to be discussed in Chapter 7. In contrast, the learning model is non-trivial though it does not tally with my current beliefs and has several outstanding but instructive deficiencies.

In principle, the learner could tackle almost any miniature environment. Although the learning model lacks the attention directing and strategic equipment needed to shift its operations from an immediate goal, it does contain a process for generating a signal indicating that there is an overload condition (problems posed under the goal it is dealing with are unintelligible) or a signal indicating an underload condition (problems posed under the goal it is dealing with can be solved, but it is impossible to construct any further procedures for solving them in different or more effective ways). The overload signal is interpreted as an instruction to build fresh procedures, since none exist (problems unintelligible) and its efforts may or may not be successful. If the efforts do prove successful the model uses the procedures it has built and it eventually reaches a condition of underload. If not, the overload condition persists and can only be resolved by a change in the focus of attention. Hence, either signal indicates an impasse unless there is some means of changing attention. A means for doing so could easily be built into the model; for example, the problem solving operations could be applied to a domain of problem descriptions (the description of a relational mesh posited in the last chapter). But this facility is not provided in the

learning model as it stands. Alternatively, the model's environment could change, by happy accident, to provide the model with the variety it needs.

In fact, the learning model is executed in concert with a teaching model which contains an attention directing mechanism represented as a tutorial strategy. This arrangement is employed to reflect the general thesis that the controller in a real adaptive teaching system (and that is the model's main field of application) acts as a surrogate for the real student's attention directing mechanism or, in so far as its instructions conflict with those of an active mental organisation, the controller usurps the position of the internal mechanism and overrides it. To represent free learning it is thus necessary to augment the learning model. But one, very restricted, augmentation is the teaching model, and the *combination* of teaching model and learning model may be viewed as a very specialised model for *free* learning.

Whichever interpretation is given (and, for the sake of clarity, only the first or 'adaptive system' interpretation is pursued) the learning model and the teaching model interact. This interaction may or may not involve the 'impasse' signal indicating a persistent condition of either overload or underload.

The teaching model, like any adaptive teaching machine, has a predictive rule built into its design, so that it can infer from the image of a student's differential performance when it is necessary to change the relation being dealt with, hence providing an environment that 'anticipates' the impasse signal and suppresses its occurrence. This alone may be sufficient to stabilise a learning process; it would be, if the prescription built into the teaching model incorporated an adequate description of the learning model or, in reality, if the prescription built into the adaptive teaching machine incorporated an adequate description of the (real) learning process. But the two components, either modelled or real, are bound to learn, in some sense, about one another, and they may get out of order. Transfer of the impasse signal is a minimal coupling which is often sufficient to bring them into register.

The hierarchical structure in the model has two levels, interpreted as levels of control (Tarjan, 1963; Mesaravic, 1962, 1963), which are henceforward designated *Lev 0* and *Lev 1*. The learning model's *Lev 0* repertoire consists in a collection of organised problem solving procedures or procedures for attaining goals/subgoals. These are called *0 operators*. Although the *Lev 0* repertoire may contain initial entries it is possible to execute the model even if the *Lev 0* repertoire is initially empty. The learning model's *Lev 1* repertoire is a collection of organised constructive processes called *1 operations*. The *1 operations* are fairly limited but there are no other than technical restrictions upon their form or complexity. In any case, the *Lev 1*

repertoire is identified with the mental organisation acquired by a real student, through maturation, imprinting, and previous learning. The most pertinent possibility is that the repertoire is programmed into the subject's mind as part and parcel of the experimental or tutorial contract (Chapter 4) which he accepts in adopting the role of student.

Both *Lev 0* and *Lev 1* operations feature as problem solvers. They differ in respect of their domain. The *Lev 0* operations have a domain of descriptions. (Of 'inputs' or of 'symbolic stimuli' presented under a specified goal relation.) Input descriptions are interpreted as intelligible problems if and only if they belong to the domain of some *Lev 0* operator existing in the *Lev 0* repertoire. If so, the learning model can begin to solve the problem by applying *Lev 0* operators; a process which may lead to correct solution or a partial solution (or a mistaken though believed-to-be correct solution). The *solution* or partial solution *is* the series of operations that are applied in a proper order. As a result of executing the solution a *response* is selected. Hence, *responses* name classes of solutions.

In contrast, the domain of any *Lev 1* operator is either the *Lev 0* repertoire or (in the case of a 'create' operation) a randomised process. A *Lev 1* solution constructs or modifies the *Lev 0* repertoire which is its internal 'response' (in the case of the 'create' operator the 'response' is the addition of a primitive *Lev 0* operator to the *Lev 0* repertoire).

From the preceding comments it is evident that the model is a normative rather than a behavioural model. That is, the experimental or tutorial situation it represents is contractual or game-like and is associated with an, albeit primitive, language in which it is possible to describe conditions and to give commands or ask questions which the real subject, in the role of student, *understands* (for example, the command 'attend to a goal relation', or a question like 'which of several statements satisfies the goal relation?'). Similarly, either the learning model or the real subject entertains an hypothesis whenever it begins to solve a problem, and this hypothesis is open to confirmation or denial (recall that a description poses a problem under the current goal relation if it is in the domain of at least one of the *Lev 0* operators in the current repertoire. But the solution to this problem may be partial or mistaken). In so far as the teaching model or the real adaptive machine delivers a signal indicating that a response is correct or partially correct, this either confirms or disconfirms the prevailing hypothesis. Any condition of the learning model that refers to a problem or its solution or the confirmation of its solution is henceforward given the general title 'state of knowing'. This rubric covers any condition upon which the model can operate or upon which it is operating.

These remarks are important because the model is exhibited in the context of the code learning task described in Chapter 8 and the 'language'

involved is so embryonic that it is easy to argue it out of existence, or at least, to consign it to a category of constructs that are pedantic enough to appear irrelevant. As a matter of history, the entire (rather lengthy) process of programming and executing the models of this family, comparing them with experimental performance, and so on, was plagued by persistent attempts, on the part of more tidy minded research staff, to 'simplify' the concepts involved. These 'simplifications', though well intentioned, reduce the family of models to triviality and reduce comparisons between the model and reality to the most banal variety of curve fitting.

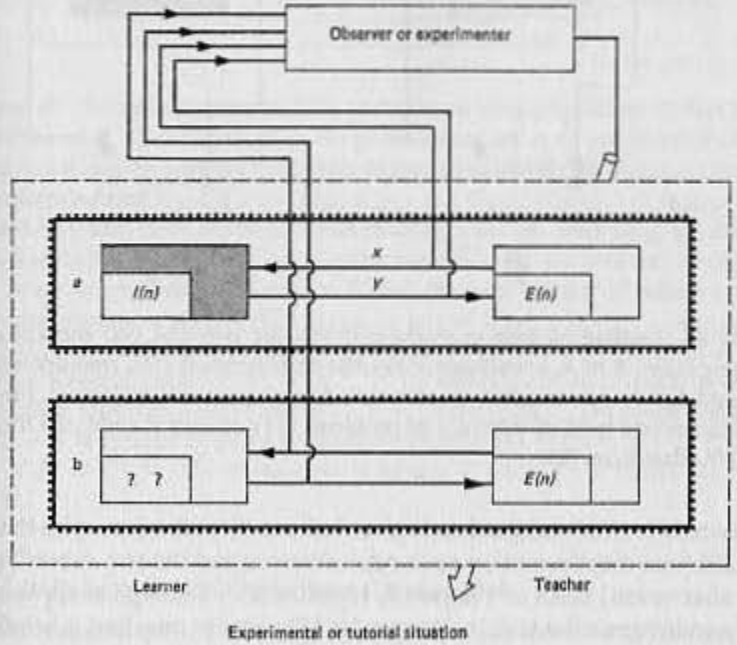


Figure 47 Summary of the family of models: $I(n)$, repertoire in learning model at nth trial; ??, repertoire of real subject at nth trial to be modelled; $E(n)$, tutorial rule together with data accumulated about the subject at the nth trial; hatched area, resource allocation and *Lev 1* operators but *not* attention direction system. *a*, simulated learning/teaching model; *b*, real learner/teaching machine (regulator).

2 Structure

The model (or family of models) is summarised in Fig. 47. A specific model is identified under a normative interpretation with a specific experimental or tutorial situation. By way of notation, the input descriptions are labelled

x, responses y and states of knowing u. The repertoire in the learning model at the nth trial in the experiment is called $l(n)$, and the tutorial rule in the teaching model, together with the data accumulated about the student, is called $E(n)$. In general the form of E is fixed throughout an experiment. But the form of E may be changed between different experiments by redesigning or reprogramming the adaptive machine.

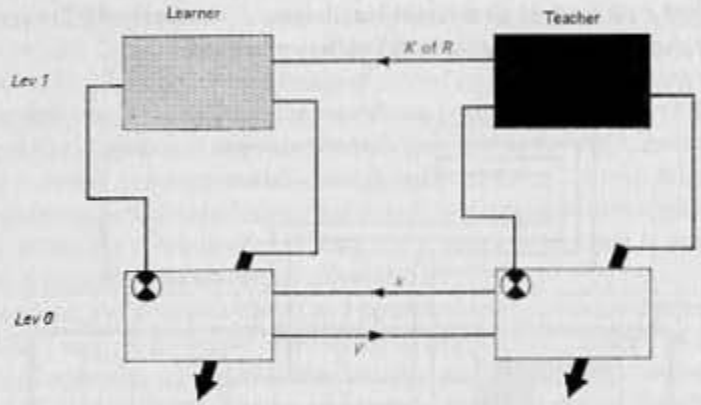


Figure 48 Outline of leaning/reaching model for one and two subskill code learning tasks: K of R, knowledge of results; cross-hatched area, resource allocation and *Lev 1* operators but *not* attention directing system; solid area, tutorial rule that selects level of difficulty of problems (x) and selects code rule for two subskill alternation tasks.

The results cited and the details given refer to identifications with the one subskill (one transformation code rule) situation and the two subskill (two rule alternation) tasks of Chapter 8, together with incomplete application to the transformation task in Chapter 10. (The reader may find it helpful to glance over the task specifications given in Chapter 8, sections 1, 2.3, 2.4 and 2.5 before proceeding.) These identifications are shown in Figs. 48 and 49. The descriptions of the subskills, used in specifying E, are isomorphic. Hence only one description is shown in Fig. 49. Certain subsets of the input sets (X_i; $X = \cup X_i$) are indexed collectively by the difficulty variable η and are referred to as sets X_η. The difficulty index (or the pair of difficulty indices η_1 and η_2) correspond (inversely) to degrees of simplification if, whenever the learning model is able to interpret an input $x_a \in X_\eta$ as a *problem* then an input $x_b \in X_{\eta-1}$ is nearer to solution than x_b and an input $x_c \in X_{\eta+1}$ is either further from solution or *uninterpretable*; that is, unintelligible. The notations, U_i U_i' stand for a set of states of knowing and some of its subsets. Thus, if there is an input x which *could* be interpreted as

a problem *if* it were presented; then if x is presented ($x \in X_i$) the 'state of knowing' is $u \in U_i$.

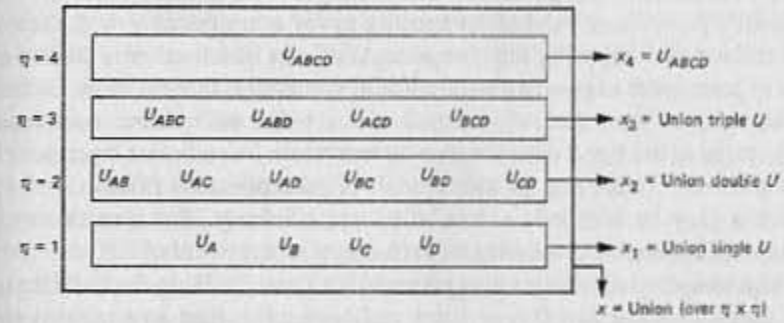


Figure 49 Stimuli are generated, as patterns, by assigning values to four signal variables A, B, C, D (2 valued in the present account, 4 valued in other experiments). η alters the *number* of variables treated as stimulus coordinates. There are 15 pattern subsets U_A, U_B, . . ., U_{AB}, U_{AC}, . . .; U_{ABC}, U_{ABD}, . . .; U_{ABCD}. For 2 valued variables there are 16 elements in U_{ABCD}; but the generating set contains only 8 stimuli, x, so chosen that if further stimuli, $x \in X$ are obtained by deleting one or more coordinates, all subsets U with the same *number* of indices contain the same *number* of stimuli (8 in U_{ABCD} or any of U_{ABC}, U_{ABD}, . . .; 4 in any of U_{AB}, U_{AC}, . . .). If X is the union of all subsets, U, then $X \subset X$. Similarly, there are four response variables A^*, B^*, C^*, D^* for which there are 15 index subsets V. A complex response, y, is a member of $Y =$ Union of all V. The correct mapping Ω; $X \rightarrow Y$ is induced by a one to one correspondence Ω^*; $(A \times B \times C \times D) \rightarrow (A^* \times B^* \times C^* \times D^*)$. A response is correct if and only if $(x,y) \in \Omega$; $x \in X$, $y \in Y$.

3 Limitations on the Operations Carried Out

In the experiments of Chapter 8 a subject is placed under a temporal restriction. If a response, y, is to count as a correct response to an input, x, the response must be made within a short interval, δt, after the input is presented. In addition, a correct response to x must satisfy $y = \mathscr{R}(x)$ where $\mathscr{R}$ is a coding rule; Ω in Fig. 49 (or, in the case of more than one subskill, $y = \mathscr{R}_i(x)$ where $\mathscr{R}_i$ is the rule in force at the trial under discussion). Chapter 8 indicates that x may be a simple input (one signal lamp) or a complex input (several signal lamps, $x = \langle s_i, . . ., s_c \rangle$) and that the number of members of x depends upon and increases with η to a maximum of 4. As the transformation rule $\mathscr{R} = \Omega$ is a coding rule, a correct response must have as many components as the input (that is, if $y = \Omega(x_i, . . ., x_c)$ then $y = \langle r_i, . . ., r_c \rangle$) but it must still be made within δt seconds after the appearance of x.

Hence, especially for larger values of η, the strict temporal constraint places a severe burden on the subject. Assuming that any mental operation whatever occupies a fixed minimum interval, it is impracticable to use some logically possible method of computing y; for example, at $\eta = 4$, there is no chance of computing the components of y in isolation. δt is chosen by the experimenter to ensure that condition. Generally, there is more latitude than this, so that the subject may *either* build up *efficient* computing structures in his *Lev 0* repertoire *or* he can retain less efficient structures in an accessible (meaning, in this model, often replicated) fashion *or* both criteria may be satisfied: accessibility and efficiency. But a modicum of trade-off between accessibility and efficiency is permissible.

The temporal constraint also restricts the time available for building up or modifying the *Lev 0* repertoire and, assuming that any constructive operation also occupies a finite interval, time is a basic resource at the subject's or at the learning model's disposal. The restriction, in this case, is imposed by the between trials interval Δt (in contrast to the solution and response constraint imposed by the lesser interval δt).

These ideas, advanced in the very specific context of one type of experiment, will bear generalisation. For any skill whatever, it is reasonable to posit an advantage in favour of efficient computation (to secure this, some resource must be expended) and to posit as well a general *resource*, available in only limited supply. This resource is needed to apply, or execute, mental operations, to build them, and to retain them as stable entities in storage.

The model incorporates this generalised concept of a mental resource which is imaged as an hybrid of data storage *capacity* and of *effort*, and which is used up in the execution of *Lev 0* operations, or the operations that build and modify the *Lev 0* repertoire. 'Storage' is conceived as a material in which operators and strings or structures of them are embodied.

It will be assumed that *Lev 1* operations in the *Lev 1* repertoire can be written and retained unchanged throughout the experiment in a costless long-term storage (not unlike the long-term memory model of Atkinson (1969), and Atkinson and Shiffrin (1967)). But there are restrictions upon the embodiment of the *Lev 0* repertoire and one way of representing them is to suppose that the prescriptions for the structures in the *Lev 0* repertoire give rise to bits of physical computing machinery which decay in time. In so far as the machinery in which the *Lev 0* structures are embodied *does* decay over time, an existing structure must be maintained or reproduced by a specifically directed activity. For this purpose a *Lev 1* operator called 'reproduce' is one *mandatory* constituent of any *Lev 1* repertoire.

This view of mentation is unashamedly drawn from the field of cellular biology. In that sense, it is idiosyncratic. With this caveat, it does no harm, to think of operations (in mentation) as analogous to allosteric enzyme

systems organised on a surface or by mutual specificity, and of problem solving transformations as analogous to the catalytic transformation of metabolites. Pursuing this correspondence one stage further, the mental computing machinery must be maintained and reproduced by a feedback-controlled, and essentially symbolic, process akin to the repressor feedback-controlled process, which involves DNA loci, messenger RNA and ribosomal transducers that reconstructs the cellular enzymes from transfer RNA-tagged amino acids, and thereby maintains the organisation in a cell. To avoid confusion we should insist that this loose analogy has nothing whatever to do with the biochemistry of brains, with the part played by messenger RNA and specific proteins in information storage (as first proposed by Hyden, 1963) or with other, equally interesting physiological issues. It is a correspondence between cellular organisation and mental organisation and that is all.

'Effort' (equivalently 'work'), denoted λ, is used up by the application of either *Lev 1* or *Lev 0* operations. However resources are allocated, there is some restriction upon the effort that can be expended or the average work that can be done in a unit interval. This effort may be spent in solving problems (by applying *Lev 0* operations), or in constructing *Lev 0* structures and in maintaining them against decay (by the application of *Lev 1* operations). Using the notion of a *Lev 1* operator 'reproduce', which maintains stored structures in the *Lev 0* repertoire against decay, and the notion that the application of this operator itself costs effort, it is possible to represent a dynamic and replicative structure which may be likened to the psychological entity 'intermediate memory' (perhaps Atkinson's (1969) short-term storage; for example, as it is organised for word-list rehearsal).

The idea of decay, together with restriction upon effort or work, leads to a system in which resources must be allocated in a very definite fashion if the computing structure is to survive; generally, its survival depends upon the introduction of at least some 'cooperative' or 'superadditive' composition rules whereby a more 'efficient' structure is more economic to maintain. The organisation responsible for allocating the resources in the learning model is a resource allocation programme.

Any resource allocation programme imposes a limit upon the average number of *Lev 1* or *Lev 0* operations that can be applied in a given interval; briefly, upon the effort, λ so that say, $\lambda_{max} \geqslant \lambda$. There is also a minimum limit, $\lambda \geqslant \lambda_{min}$, which reflects the fact that operations must be applied to something, though not necessarily to the problems that are deemed to be relevant by the experimenter.

Many resource allocation programmes satisfy these conditions. For example, the resource allocation executive of the model in Figures 50 and 51 satisfies:

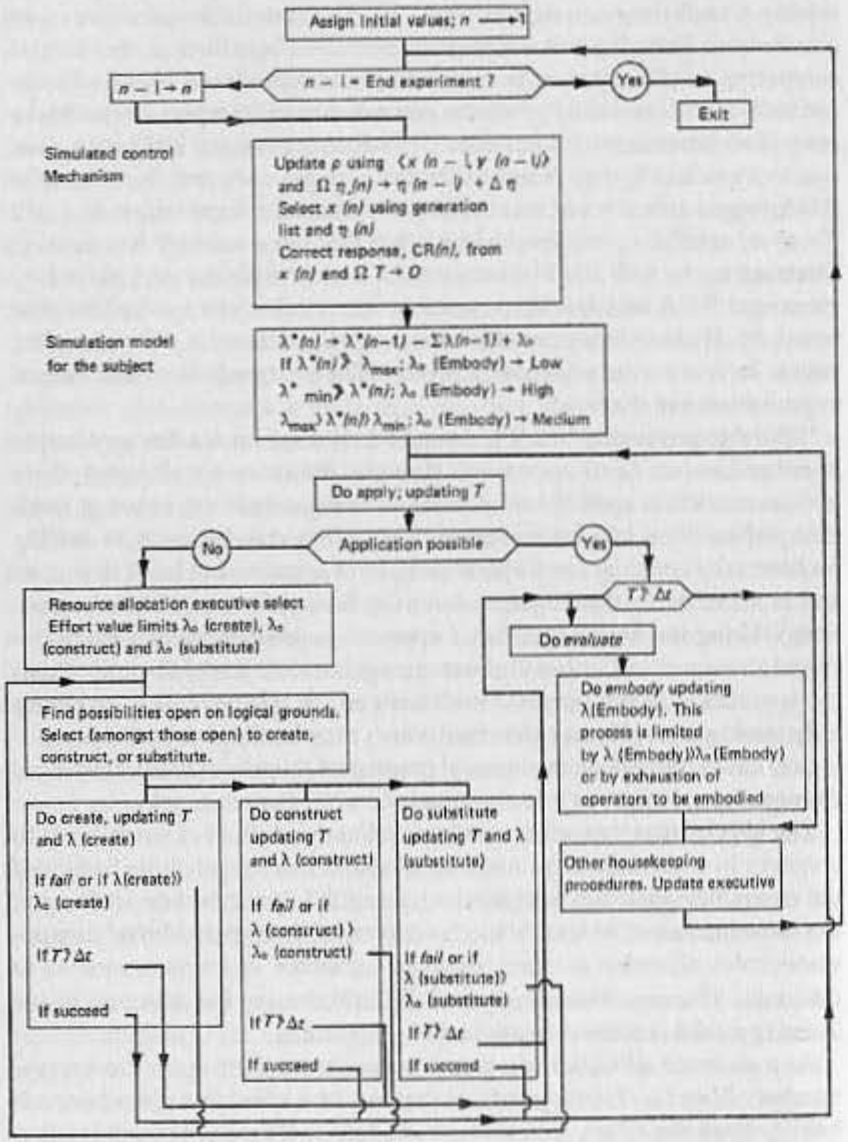


Figure 50 Overall organisation (simulated control mechanism and model for subject. Flowcharts for a single subskill version of the model (due to Feldmann, Lewis, Mallen and Pask) current in 1966–8.

(*a*) Problems x numbered as *SN*; 1–8, $x \in X_1$; 9–32 ,$x \in X_2$; 33–64, $x \in X_3$; 64 and above, $x \in X_4$. The program *decomposes* the problems into their constituents by rewriting rules based on the description in Figure 49.

(*b*) Response *components* of y designated by *KP*.

(*c*) Operators have *domain* and *range*; simple operators domain *SN* 1–8; *complex* operators, obtained via substitution, a multiple domain.

(*d*) *Range* is member(s) of *KP* (since Ω is one to one, any correct operation has domain of η elements).

(*e*) *Concatenation* produces *strings* via process of string *construction*.

(*f*) *Domain* of string: $SN > 8$.

(*g*) *Range* of string; set of operator names.

(*h*) Operators and strings have *lifespan* (trials to entry of a deletion stack) and a *utilisation* (their correct use record).

(*i*) Various stacks are used in program but two are distinguished as *operator store* and *string store*.

(*j*) $\lambda^*(n)$ is effort available at trial n.

(*k*) T is time taken; Δt is time permitted.

Notes on the main variants.

(*a*) Predictive control of effort using estimate of $\Delta\lambda^*$.

(*b*) For double subskill (or interference task) model there are two stimuli $x = x_A, x_B$ of which the control mechanism gives priority to x_A. *Other* subskill (interfering task) is assigned value, like a procedure, and x_B is selected for attention in the same way that procedure is selected.

On following pages: *Figure 51* Flowcharts showing details of model organisation: (*a*) *apply* operator procedure; (*b*) *create* operator procedure; (*c*) the most elementary type of concatenation (string construction) procedure ($x \in X_2$); (*d*) substitution procedure for constructing complex operators from existing strings and for replacing these strings; (*e*) outline of *Evaluate* response procedure and the *Embody* procedure (for updating *lifespans* so that operators/strings are retained in operator store/string store). Many variants were used. (*f*) Outline of resource allocation executive controlling the (relative) effort, λ assigned to different processes.

Figure 51(a)

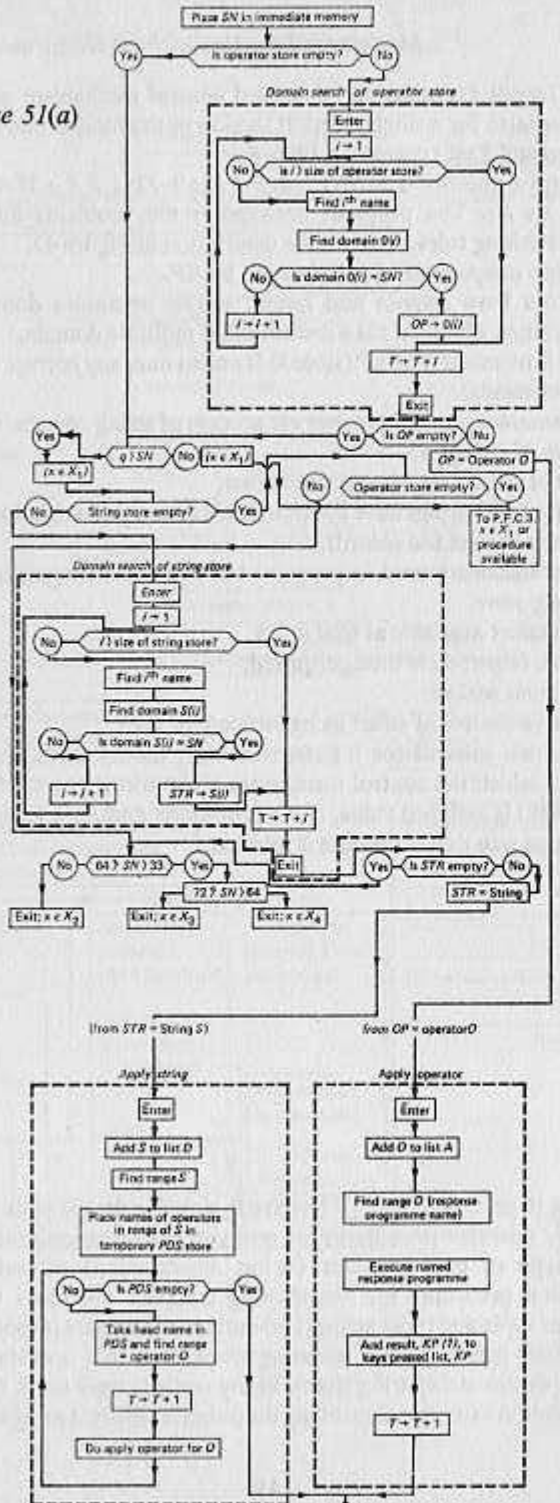


Figure 51(b)

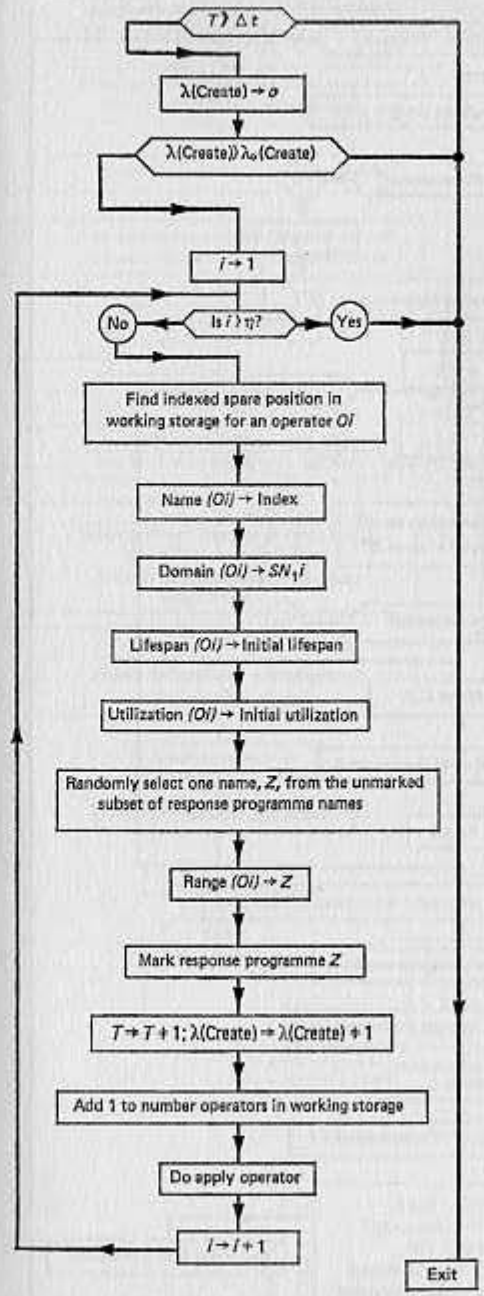


Figure 51(c)

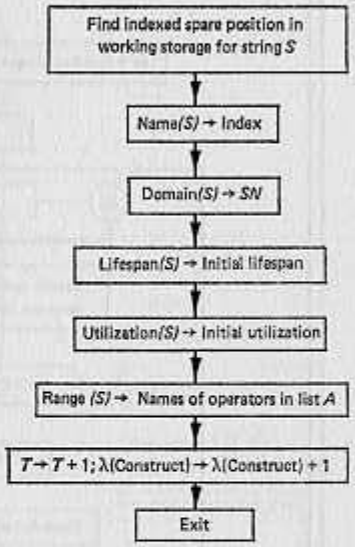


Figure 51(d)

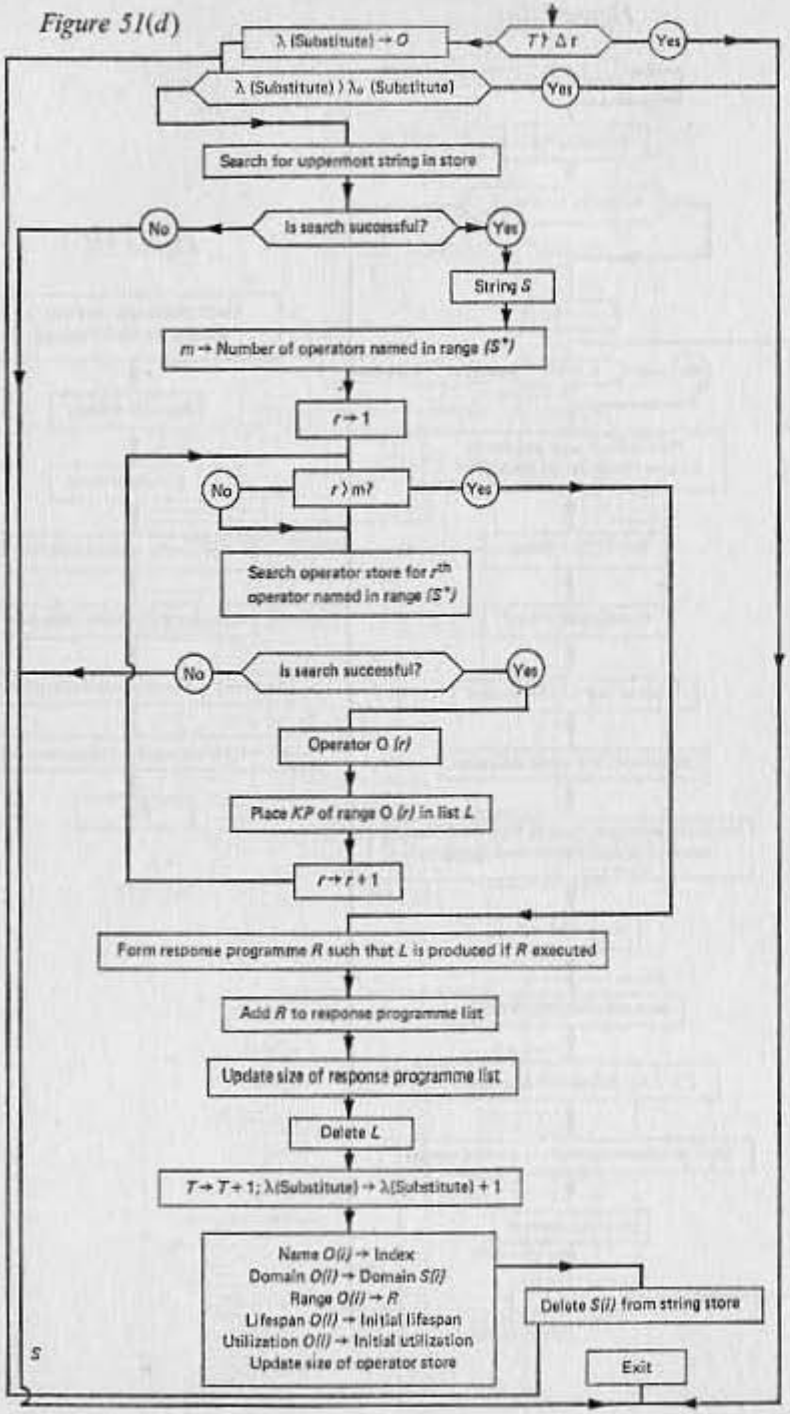


Figure 51(e)

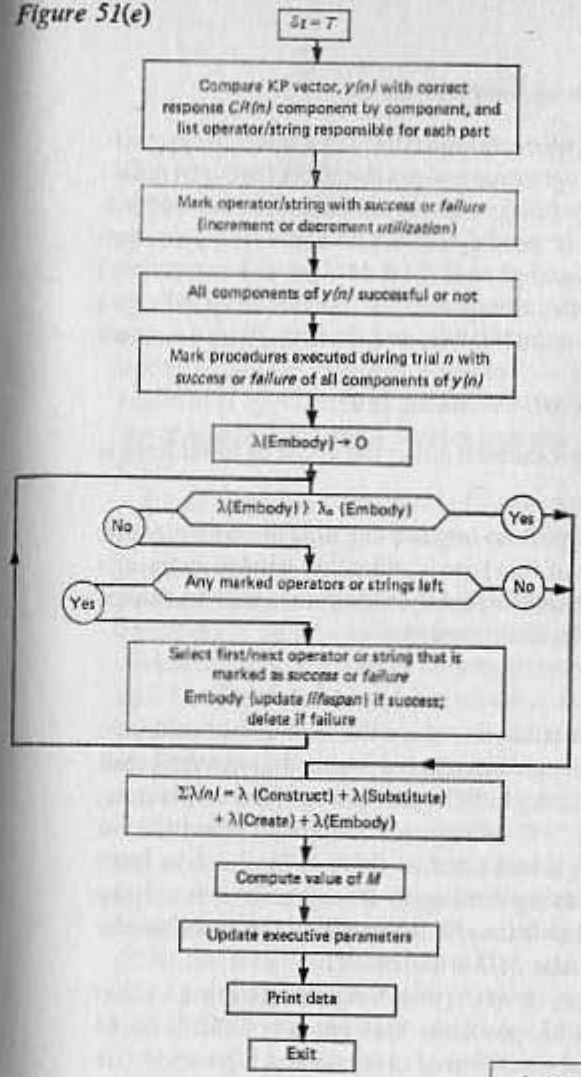
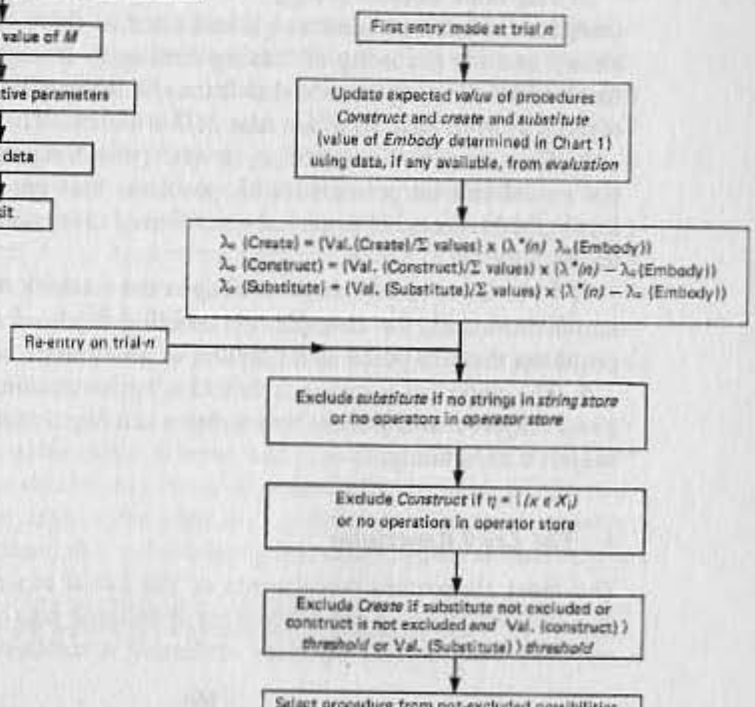


Figure 51(f)



1. The average value, λ^* of λ is constant, so that $\lambda_{max} > \lambda^* > \lambda_{min} > 0$.

2. The resource allocation programme assigns the effort (between different sorts of *Lev 1* and *Lev 0* operations) so that (a) at least some *Lev 1* operations are applied to construct or modify strings of embodied operators (hence, there is at least some learning) and, (b) if MU (model uncertainty) is a variable that increases whenever problems remain unsolved, that increases rapidly if a problem is unintelligible, and that otherwise decreases then

$$MU_{max} > MU^* > MU_{min} > 0$$

and an 'impasse' signal is being produced if either the upper or lower limit is contravened.

If a learning model able to interpret an impasse signal is governed by any resource allocation programme of this type it will either behave as though it was capable of curiosity or it will, if slightly elaborated, tend to change its attention and to reinterpret its environment.

In particular:

1. Learning is a process that tends to reduce the machine uncertainty, MU (interpreted as an uncertainty given a class of problems, regarding their solution) by applying and if necessary building structures of *Lev 0* operators.

2. The limit $MU > MU_{min} > 0$ reflects the empirical condition of underload. If this requirement is not satisfied there is 'nothing to learn about' and the absurdity of 'having nothing to learn about' can only be resolved by allowing the model to learn about something the experimenter regards as irrelevant; in which case MU is undefined.

3. The converse limit $MU_{max} > MU$ (which represents the fact that the model can be presented with problems that are too difficult to be intelligible) reflects the empirical condition of overload and high subjective uncertainty.

4. In order to satisfy these conditions the teaching model must control the environment, for example, by selecting the level of difficulty of the problems that are posed as a function of an estimate of MU.

5. The empirical correlate is that an adaptive teaching machine controls a real subject's environment by varying η as a function of its estimate of the subject's uncertainty.

4 The *Lev 0* Repertoire

The most elementary constituents of the *Lev 0* repertoire are simple *0 operators* which carry one component of an input into one component of a response. Any correct operator mediating a solution, carries the input

component into a correct response component. But an *0 operator* may, just as easily, lead to a mistaken response.

The next constituents are complex *0 operators* that carry pairs, triples, quadruples, etc., of input components into pairs, triples, quadruples, etc., of response components. Whereas simple *0 operators* might be applied (given appropriate tests and checks) to yield the correct response $y = \langle r_1, \ldots, r_c \rangle = \Omega(x)$ to a complex input $x = \langle s_1, \ldots, s_c \rangle$, a complex c-fold ($c > 1$) *0 operator* achieves this result at once. Both simple and complex *0 operators* can be created *de novo*; in essence, by guessing, but the chance of creating a correct complex *0 operator* in this manner is very small.

Finally, there are structures of *0 operators*, called *0 operator strings*, which are entities containing the addresses of *0 operators*. A c-fold *0 operator string* is applicable to an input $x = \langle s_1, \ldots, s_c \rangle$ for $c > 1$ and musters the simple *0 operators* needed to respond (correctly or not). A string and the operators it addresses may both and independently be mistaken; hence, mistaken parts of structure may be providently winnowed out. Strings may also be substituted to yield equivalent complex *0 operators* that solve the problem posed by $x = \langle s_1, \ldots, s_c \rangle$, at once. Using the string intermediary (if the resources are available to do so), the model is almost certain to build a complex *0 operator* that leads to a correct response.

The stringing operation is generalised to yield strings of strings[1] (and so on) and to yield strings of complex *0 operators*. A string of complex *0 operators* is also open to substitution, conversely, a string of one *0 operator* is counted as a *string*. With these interpretations, the data structure $l(n)$ of the *Lev 0* repertoire is shown for trial $n = T$ (the trial at which the skill is learned and performed with complete proficiency) in Fig. 52. The caption relates $l(T)$ to the previous discussion of problems and solutions.

Although *0 operators* are specified in a very simple fashion for the coding skill, the concept of $l(n)$ is open to an arbitrary degree of elaboration. For example, the simple *0 operators* are quite easily replaced by small finite-state machines (Fogel, Owens and Walsh, 1966; Pask *et al.*, 1968) in which case a string becomes a program of FSMs, and a complex *0 operator* a composite or restricted product, of FSMs.

Each *0 operator* or *0 operator string* in the *Lev 0* repertoire, $l(n)$, is associated with two variables called lifespan and utilisation. Both variables are used to simulate processes that could be realistically executed. Of the two, lifespan is given an initial value when an *0 operator* or a *String* is embodied in the *Lev 0* repertoire, $l(n)$; subsequently it is decremented at each step in

1. The notion of 'string of strings' was originally introduced by Mr G. L. Mallen who was responsible (see Tech. Rep. references) for programming one of this family of models.

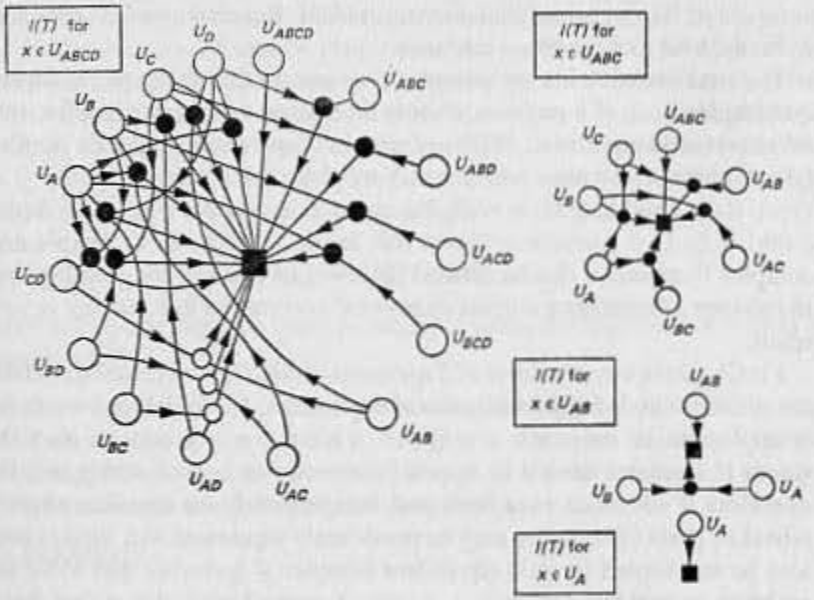


Figure 52 Each input, x, gives rise to a state of knowing because it is a member of some set U. Sets U are shown as nodes. A *problem* is posed by x, under rule Ω and is solved by applying a correct operation, a string of operations, or a complex operation (in turn, yielding other states of knowing). *Correct* operations are shown as directed arcs, which lead to a unique and central node (the *solution*); their application produces components of a response, y. The *description* of the correct operation paths is $I(T)$. To simplify the diagram $I(T)$ is shown for $x \in U_{ABCD}$ (representing $\eta = 4$) for $x \in U_{ABC}$ (representing $\eta = 3$) for $x \in U_{AB}$ (representing $\eta = 2$) and for $x \in U_A$ (representing $\eta = 1$).

the execution of the model to simulate decay and the operator or string is deleted from $l(n)$ if its lifespan becomes zero. Deletion of a string does not imply deletion of the *0 operators* it addresses nor does deletion of all the addressed *0 operators* imply the deletion of a string. Conversely, the value of lifespan is incremented if the operation *reproduce* is applied to the *0 operator* or string in question to simulate its replication. Utilisation is incremented whenever an *0 operator* or string is applied and its value is decremented, to a minimum of zero, for each trial.

5 An Account of the *Lev 1* Repertoire

The first *Lev 1* operation is APPLY the shortest applicable string of *Lev 0* operators in $l(n)$ to the problem denoted by an input. APPLY is only realised if such a string of operators exists in $l(n)$. (For example, for problem *a*

(Fig. 53) when it leads to the application of a single *Lev 0* operator or for problem *b*, when it leads to the application of a string of operators. APPLY is not realised for *d* or for *c*.) In so far as APPLY is realised, it is given precedence and it leads to a solution of a problem and to the partial satisfaction of the goal. If the model is presented with a problem such as *d* or *c*, the model is unable to APPLY and such a problem is interpreted as a sign invoking the *Lev 1* operations described below.

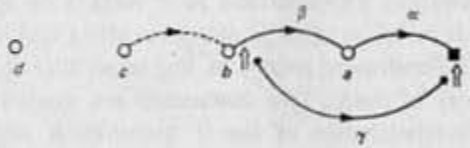


Figure 53 States of knowing (arbitrarily chosen ones) shown as nodes *a*, *b*, *c*, *d*. Operators are shown as directed lines, α, β. The dotted line indicates that concatenation onto the end of string (α, β) is possible, thus interpreting *c*; however, the chance of randomly constructing an operator to interpret *d*, though finite, is small. Double arrows on operator γ indicate substitution process.

The first of these is CONCATENATE. To CONCATENATE, the model searches an operation list for a *Lev 0* operator which may be applied to the problem posed by an input, or part of it, and which may also be adjoined to a string of operators in $l(n)$ which terminates in a solution state. The gap which the concatenation process is allowed to span is restricted (essentially, by the length of list search and the resource allocation programme). In Fig. 53 it is assumed that the gap between *b* and *c* allows for concatenation although that between *b* and *d* is too great. The model would have a vanishingly small chance of randomly CONSTRUCTING an operation which leads from *d* to *b* (and thus to *a* and the solution) within the interval allowed for solving the problem.

CONCATENATE is always followed by a *Lev 1* operation EMBODY the selected *Lev 0* operation as an operator in $l(n)$. Further EMBODY is always followed by APPLY and the persistence of the string of operators formed by CONCATENATE is indirectly dependent upon the receipt of a 'knowledge of results' signal (the result of an external test) which evaluates the application of the concatenated string of operators. Since concatenation is controlled by 'knowledge of results', $l(n)$ chiefly consists in strings of operators that lead to correct solutions.

The next *Lev 1* operation is SUBSTITUTE. The model performs this operation, if the resource allocation programme allows it to do so, whenever it has completed an application or has found that application and concatenation are impossible. Substitution consists in the placement or embodiment in $l(n)$, of a *complex 0* operator performing the same (or in principle a

similar) job to an existing string of simple *0 operators*. Substitution using a complex *0 operator* leads to the construction of shorter and more efficient *Lev 0* structures. The basic substitution process is exemplified in Fig. 53 by the placement of γ in parallel with the string of operators, α, β. Since APPLY selects the shortest applicable string of operators γ will be used upon subsequent occasions and α, β, will not be used. As below, this may lead, indirectly, to the decay of α and β.

The *Lev 1* operation REPRODUCE is a variant or specialisation of SUBSTITUTE. It is called as often as resources allow and 'reproduces' any *Lev 0* structure (operator or string) in the sense that it substitutes the original by a copy of itself. Two comments are needed (a) a string is REPRODUCED independently of the *0 operators* it addresses and (b) REPRODUCE is simulated in the learning model by an incrementation of lifespan which is decremented, each unit interval, to simulate *decay*.

For the two subskill tasks where there are *Lev 0* repertoires $l_1(n)$, $l_2(n)$ the *Lev 1* repertoire is augmented by operations akin to substitution that carry out transfer of training; they take parts of $l_1(n)$ and place them in similar positions in $l_2(n)$ or vice versa.

Most versions of the model (including the version shown in Fig. 50 and Fig. 51) incorporate a process for attending to and analysing detailed knowledge of results (i.e. data regarding the rectitude of parts of a response obtained by comparing these parts with parts that are already established or a component-specific corrective input). In the case under discussion, though not generally, the cost of applying this operation is zero.

Finally, implicit in the SUBSTITUTE operation, there is an operation, DESCRIBE. In order to substitute, the model must know what operator or string of operators in $l(n)$ is similar to an operation on its operation list. To do so, it must describe the contents or some of the contents of $l(n)$. The elements that are described are coded like members of the operation list, and their similarity with other members is determined. Hence DESCRIBE is an operation for adjoining members to the initial operation list (for example, the coded description of the string of operators α, β, which turns out to be similar to γ).

Notice that the derivation of γ in Fig. 53 may either be conceived as the matching of an existing code 'γ' with the description 'α, β', or as the production of γ from α, β, by a rule applicable to descriptions of operator strings, ostensively defined by embodiment in $l(n)$ (recall that most of the strings of operators in $l(n)$ lead to correct solutions).

The execution of any operation decreases the amount of effort, λ, that is available. Conversely no operation can be applied (or even instituted by the resource allocation programme) unless the necessary effort *is* available. Effort is incremented at the conclusion of a trial and may be saved or carried over from one trial to the next.

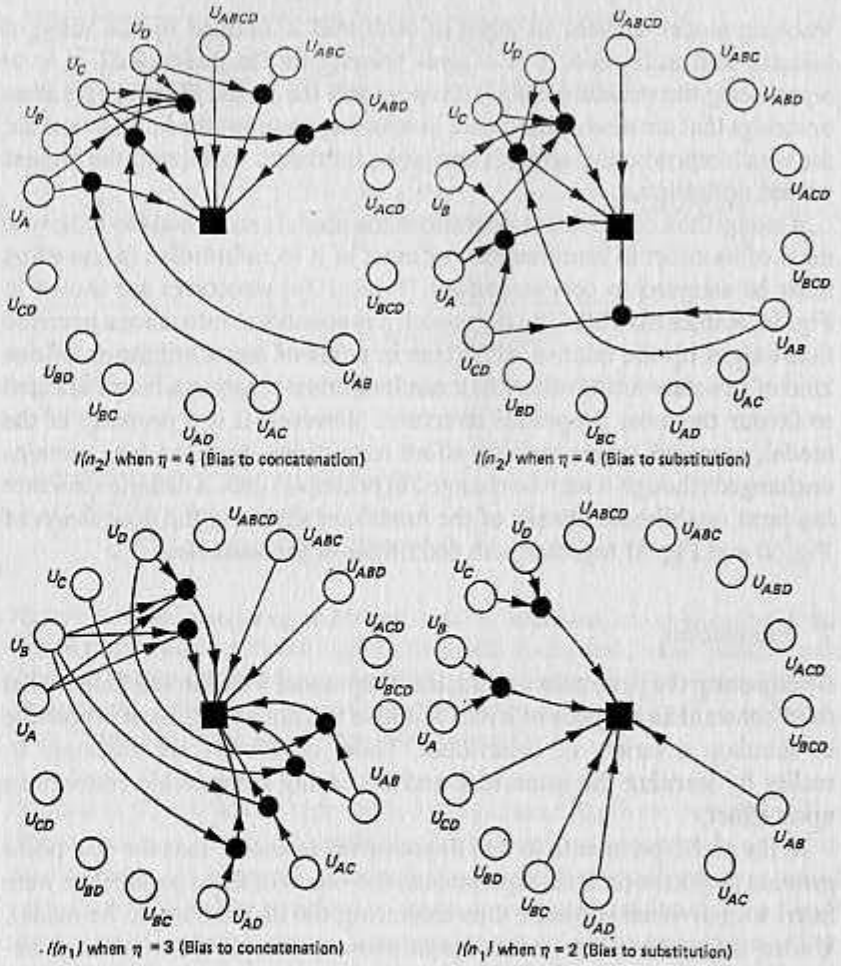


Figure 54 Typical structures of $l(n)$ shown on left for one successful mode (bias to concatenate operators) and on right for another mode (bias to substitute operators). In each case $n_2 > n_1$ and the sequence is associated, realistically, with increase in η as learning proceeds; with construction of some complex operators and deletion of others. The model does not (run normally) have the effort to use and apply the processes that muster four or even three operators. Further, unless redundant complex operators are deleted (shown in Bias to substitution case) the model does not have effort to embody essential operators.

6 Operation of the Learning Model

In the interval δt, or until a response is produced within that interval, the resource allocation programme gives priority to the *application* of *Lev 0* operators and strings in order to solve the problems posed when the

teaching model delivers an input at each trial n. Guided by the value of lifespan and utilisation it also gives priority in the interval $\Delta t - \delta t$ to *reproducing* the structure in $l(n)$; it reproduces the lowest lifespan operators or strings that are also utilised and, in selecting amongst the limited number for which reproductive effort is available, it favours those with the highest valued utilisation.

Amongst the constructive operations the model can be biassed to devote most of its effort to *concatenation* or most of it to *substitution* (some effort must be assigned to concatenation). Typical $l(n)$ structures are shown in Fig. 54. Rather than biassing the model, it is possible to introduce a heuristic that weighs up the relative advantage in terms of mean utilisation of one kind of structure or the other. As a result of this evaluation a bias is selected to favour the most propitious structure. However, it is a property of the model, executed under realistic effort restrictions, that the bias remains unchanged (though it *may* be changed in principle) once a definite structure has been established. Details of the model are shown in the flow charts of Fig. 50 and Fig. 51 together with definitions of the variables.

7 Simulation

By adjusting the parameters of the learning model with the teaching model fixed constant as a replica of a real adaptive teaching machine, it is possible to simulate a variety of behaviours. Those of interest are anchored to reality by learning the same task and by having comparable restrictions upon effort.

In the real experiments Δt and δt are varied to ensure that the task poses genuine problems (in later experiments, the values of these parameters were fitted to individual subjects, thus enhancing the distinctions to be made). Under these conditions, all subjects show a sharply discriminated behaviour (a) in terms of their susceptibility to overload and (b) in terms of the distribution of complex response component latencies. The latency patterns serve as an indication of the problem solving method (taking individual subjects and examining response to each input separately, there is an all or none distinction between the learning patterns referred to as stringing and grouping in Chapter 8). The former (real-life) pattern corresponds to an $l(n)$ (in the learning model) biassed to concatenation (Fig. 54) and the latter to an $l(n)$ biassed to substitution; moreover, if the relative application latencies of the model are recorded, these closely replicate the real-life response latency profiles.

It is still possible to simulate many behaviours by adjusting the learning model parameters. Taken out of context such exercises have doubtful value. Consequently one phenomenon has been isolated as worth illustrating

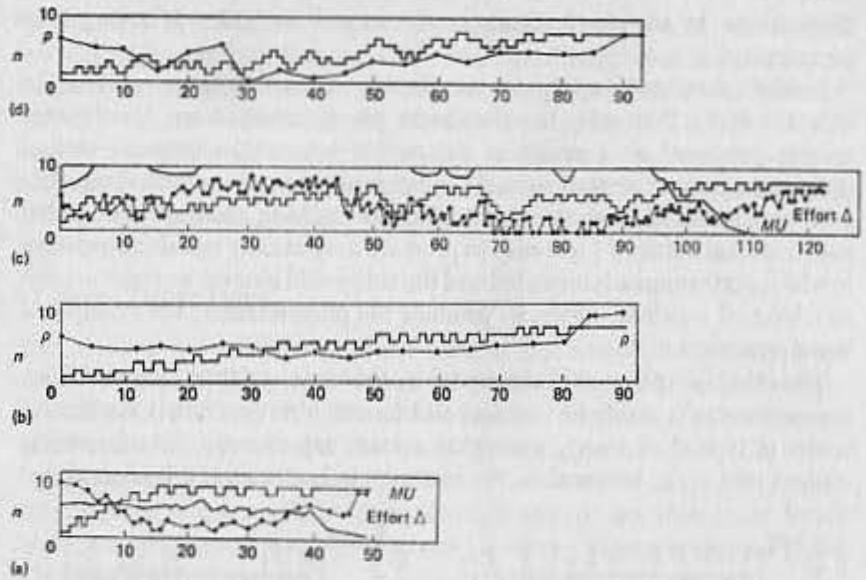


Figure 55 Data comparing stable and instable behaviour in a subject and in the learning and teaching model: (*a*) stable model (same task, same parameters); (*b*) stable subject (same task, same subject); (*c*) instable model (condition induced by overload); (*d*) instable subject (condition induced by overload). ξ = Steady State Level; ρ = Proficiency Index (see Chapter 8).

because it is particularly repeatable and is interesting in its own right. The phenomenon is a form of instability due to overload (apparently any kind of overload; for example, overload by an auxiliary task or by a small reduction in δt). The overload is most conveniently induced by a small reduction in a parameter ξ, common to the adaptive teaching machine and its veridical image, the teaching model, without an alteration in the learning model parameters. Some data are shown in Fig. 55 which is a comparison of stable against instable behaviour in a subject, and stable against instable behaviour in the learning and teaching model. The real system characteristics are typical.

The instability is manifest (Fig. 55) as an oscillatory fluctuation in η and a departure of ρ from its steady-state value. The curves for the learning/teaching model and for the subject/machine system are very similar in form. The course of the instability is evident in the model only; a periodic depletion of effort (recorded as an effort surplus, Δ, rather than the effort level, λ). Moreover, subjects who give a retrospective account of their experience consistently report periods of high uncertainty alternating with periods where they have a grasp of the task. It is tempting to liken these

fluctuations to the fluctuations in the model variable, *MU* (machine uncertainty).

Similar comments apply to the double subskill system outlined in Chapter 8. In this case, the dominant phenomenon is an interference barrier produced as a result of destructive interaction between mental operations proper to each subskill, which must be surmounted if the skill is to be learned as a whole. If the adaptive teaching machine is operated near a critical value of ξ it is easy to produce a repeatedly instable condition in which performance is impaired and the entire skill is never learned; almost any kind of overload serves to produce the phenomenon; for example, a small reduction in δt.

The effect is replicated in the model by the same variation. Figure 56 is a comparison of a stable but critical and an unstable (overload) simulation which is typical of many, somewhat varied, experiments. Interference is evident (the η_1, η_2 interaction, for example) in both cases; it is engendered

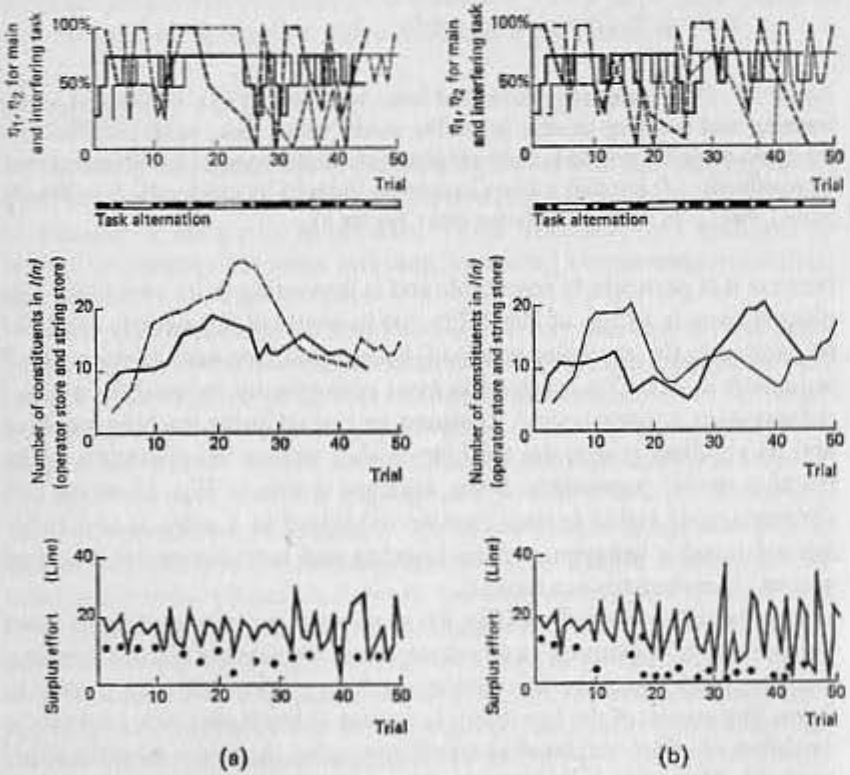


Figure 56 Comparison of: (*a*) stable and (*b*) instable simulation for double subskill conditions.

by the task specification and is unavoidable. However, no significant learning takes place in the unstable case. The direct cause is a failure to accumulate sufficient common structure in $l_1(n)$ and $l_2(n)$ (solid line and dotted line), one structure in the unstable model ($l_1(n)$) is abraded whilst the other structure ($l_2(n)$) is built up and vice versa. This degradation can be traced back to an indirect cause, namely a periodic depletion and oscillation of surplus effort evident (lowest graph) in the unstable case only.

8 Some Other Data

The learning model has also been used in connection with the type of situation noted in Chapter 10 where the task is isomorphic with the single subskill code learning task. The environment is partitioned to permit the specification of several distinct learning strategies of which the most important are jointly characterised by an incidence of 'strong' response latency patterns and a serial learning strategy, and (on the other hand) by an incidence of 'grouped' response latency patterns ('clump responses') and a holistic learning strategy.

Thus employed, the model demonstrates the salient distinctions in the experiments, not so much in terms of replicating the statistical results but by furnishing a mechanism to explain the empirical differences, quite generally. The findings are summarised as follows.

1. An effect, later called *cognitive fixity*, leads to entrenchment of a structure in $l(n)$; either the bias to concatenate structure or the 'bias to substitute' structure of Fig. 54. These structures are incompatible in the sense that effort spent in replicating one structure fails to replicate, or even degrades, the other structure.
2. A 'bias to concatenate' structure leads to string response latency patterns.
3. A 'bias to substitute' structure leads to grouped or 'clump' response latency patterns.
4. A 'bias to concatenate' organisation can be induced (in an otherwise unbiassed learning model) if it is driven by a serial teaching strategy (Chapter 10).
5. A 'bias to substitute' organisation can be induced (in an otherwise unbiassed learning model) if it is driven by a holistic teaching strategy (Chapter 10).
6. If the learning model is initially biassed by any means and it is instructed by a teaching model equipped with the converse teaching strategy, learning is inhibited and the model's activity closely resembles the empirical phenomenon of *mismatching*.

9 Criticism and Extensions of the Model

Learning and teaching models of this kind are readily made more complex and able to carry out cognitive operations that deserve the name 'problem solving'. For example, the simple *Lev 0* repertoire can be filled out with a program suite such as Newell Shaw and Simon's *General Problem Solver*, most succinctly reported in Ernst and Newell (1969), by a very different process such as Quillan's (1969) *Teachable Language Comprehender* or by any of the artificial intelligence programs reviewed in Minsky (1968). As an intermediary step there are many more specific programs such as those reported in Hunt, Marin and Stone (1966), Reitmann (1965), or Feigenbaum and Feldman (1963). By the same token the *Lev 1* repertoire can be filled out with operations for rewriting and amending these programs (the more elaborate exemplars, such as the *General Problem Solver* and *Teachable Language Comprehender* have these constructive operations already incorporated).

The distinction between *Lev 0* and *Lev 1* models is best made by programs that do incorporate such a facility. For example, Pereira (1973) has established a heuristic serial procedure for problem solving of a very general kind based upon an information measure (Chapter 1).

There are, however, several outstanding deficiencies which are more obtrusive if the simple model is kept in mind and are not eliminated simply by making it more *complex*. The specific elaborations of the last paragraph do not remedy the defects and an arbitary increase in complexity, *per se*, is utterly ineffectual.

1. Certain processes which might be executed are in fact imitated. The outstanding example is *reproduction* which, in this model, is sham (quite deliberately so, for the sake of computational efficiency). It *need* not be a sham, of course (Chapter 3), and it is vitally important that it should not be if the model is to penetrate further into the nature of cognitive processes. For, on seriously enquiring what a concept *is* or an intention (i.e. a goal) *is* or, first and foremost, what a memory *is* one answer turns out to be 'they *are* species of reproductive process' (in the sense of the theory of 'self reproducing automata' and its derivative disciplines). A memory, for example, is a procedure that reproduces a concept (itself a reproductive procedure); a point of view that is developed in Chapters 10 and 11.

Insightful readers may have noted that *Lev 1* operations and *Lev 0* operations figure as the components in one of Von Foerster's finite-function machines; the *Lev 1* repertoire acts upon the state sets of the 'machines' in the *Lev 0* repertoire. By a mild but useful extension of Von Foerster's theory the *Lev 0* operations are conceived as automata (not

necessarily finite) and the *Lev 1* operations as reproductive automata that reproduce them, possibly with variation, mutation, recombination and the like, to generate an evolutionary process called *learning*. The point is that such a process is *imitated*, to an arbitrary degree of complication, by the model under discussion. With certain important caveats such a theory is *executed* by the models discussed in Chapter 3.

2. A process of this kind inhabits a very special environment *in* which it is reproduced. One candidate is a relational network (mooted earlier) together with a description of the related relations in terms of *what may be done* to bring them about or satisfy them and *what may be known* or learned (how one relation can be constructed by applying suitable operations to several others, also in the network). The whole notion of *knowability* or *memorability* is glossed in the present model (the reader may have felt uneasy over the inherent memorability of relations like $\mathscr{R}_1$, $\mathscr{R}_2$, that determine interfering subskills). To remedy this defect, canons of the following kind are mandatory. A domain of relations is knowable if (a) the relations in question are brought about by reproducible *Lev 0* structures, (b) there is a description scheme which allows the model to construct and reproduce them, and (c) each relation can be instanced in some world (for example, by constructing an artifact, doing an action, or writing a computer program that is executed).

Of these requirements (a) and (b) are commonsense restrictions that have recently been given a formal stature. (c) appears, at first sight, somewhat different in kind. It is not. The basic requirement is executability; a correct solution to a problem is a solution that can be executed to bring about or stabilise a relation.

3. The learning which is modelled takes place as part of a learner/teacher interaction. In the model as it stands, the learning and teaching parts are arbitrarily separated by two constraints (those of the last chapter), that stem from *imitating* rather than *executing*. The first constraint is the imitation of language rather than its realisation. It would not be difficult (even in the present framework) to make a better imitation; that is, to convert the interaction into a transaction scheme involving a formal *syntactic language* where (as in Chapter 4) a command form is taken to be interpreted as a command and a question form as a question, for metatheoretic reasons (the language game, the communication game) that are beyond the formal scheme. More than this is needed. As a minimum a *genuine* (not imitation) language involves command and question transactions that are *addressed* to pronominalised recipients for purposes (the pragmatics of the language) and the consequences of which are interpreted (the semantics of the language).

Such a language can be (and recently has been) provided in the real system, i.e. the mechanical 'teacher' can be furnished with the equipment to talk thus to a real subject who has the purpose of 'learning about' the same environment (Pask and Scott, 1973; Pask, 1973b). An immediate consequence is that nearly all operations (for example, the chains of explanation that are the linguistic forms of reproduction) become distributed, in the sense that parts are executed in the 'learner' and parts in the 'teacher' ('distributed' execution, as in Chapter 5).

4. The second constraint that comes from imitating rather than executing, is closely related to the doubts and difficulties aired in Chapter 5. The learning model and the teaching model are synchronised by edict, so that the order relations amongst the various processes is dependent upon factors outside the model itself. It is generally true that any systematic operation does entail synchronisation (or partial synchronisation). But, in real teaching and learning, the synchronisation is not built in from the outside. Learners and teachers are substantially asynchronous apart from the communication they engage in; they *become* synchronised *because* of it, or, viewed conversely, the communication (information transfer in Holt's (1968) and Petri's (1965) sense of the word) is a symptom of local synchronisation.

No real 'human brain' versus 'computing machine' issue is involved. A machine (though not a standard machine) can execute the requisite procedures. A brain which is a non-standard biological machine *does* so. But would the operation of a working 'model' that executes these procedures be deemed a model or a reality? It is not in any ordinary sense, a *simulation* though its operation may be 'simulated' by dint of the gambits (splitting processes into short segments, storage of intermediary results and random selection of one member from a set) that are usually employed to regularise partial synchronisation of fuzzy processes.

5. Systems of this type operating in the domain of intellectual learning are considered in the next volume and introduced towards the end of Chapter 10 and in Chapter 11. Very similar work has been done independently by Dexter (1972) in concert with Seigmann.

But, as promised the essential properties are captured by motor learning no more sophisticated than the transformation/coding/skill. For example, Scott (1963) has written a stripped-down model for a 'one handed' typist learning both *concepts* such as keyboard descriptions and *operations* like complex finger movements; it is a fairly elaborate piece of programming because the reproductive operations are genuine (not faked, as they are in the model under discussion). As a result, the model demonstrates the phenomena (such as the establishment of description schemes, their disappearance

from awareness as the skill becomes automatic and the motor operations are executed by many processes acting in parallel) that were observed during a lengthy and detailed investigation of teleprinter operator training (Pask, *et al.* (1969a) or Scott and Pask (1973) in press).