10 Processes and Prescriptions for Action and Learning

This chapter describes some problems of human performance and learning from a process-oriented viewpoint. The concepts stem, in part, from studies of industrial and commercial organisations and from studies of the teaching process in the areas of programmed learning and computer assisted instruction.

Section 1 deals with prescriptive, descriptive and permissive structures made up from commands to *do*; section 2 with rather narrow prescriptive structures made up from commands to *learn*, and section 3 with a detailed case history. The theme is continued into the next chapter on strategy in which it is possible to introduce permissive and descriptive structures and to voice a need for very general representations of the *learning* process.

1 Process Instituted by Commands to *Do*

In various branches of industry, processes such as clerical and inspection routines or even the operation of a whole department are represented by flow charts, decision tables, and condition charts (including critical path and PERT meshes). These instruments are used descriptively to give an account (e.g. in work study) of how people do jobs; prescriptively, to induce or monitor performance and, in the case of the meshes, permissively. (A flow chart depicts one process; variations are induced by external conditions only).

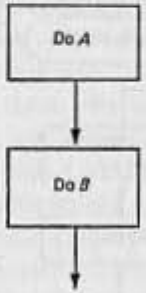
It is provident to concentrate upon the prescriptive application as the most frequent and successful one and to comment, where necessary, upon description and permission-giving. In this case, the addressee (the executive agent) is a human being and the commands (including questions) are humanly interpretable. Throughout this section all commands are commands to *do*, but it is possible that an addressee who is unable to obey these commands will interpret some of them as commands to learn how to do.

There is no evidence of a strictly behaviouristic identification of the type which is very common (section 2) in the context of commands to *learn*. The idea that a *prescription* for a process might be a series of semantically-

neutral stimuli evoking (causally-related) responses has little currency and is pretty obviously untenable against the background of a process governed by rules and regulations.

1.1 Instruction Sequences Any linear arrangement of unconditional commands (such as a check-list or a rudimentary manual) may act as a process prescription, on the assumption that the addressee can interpret the commands and that he is able, as well, to interpret the order in which they are written (that is, a serial order; man is asked to use his brain as a serial processor). Such a prescription can be executed but is only of value in very static situations, where its execution is likely to produce the desired result only because the environment or input data is *not* variable. If the prescription is regarded as a prescription for control and the executive agent as a *controller*, then he is a *feedforward* controller and is liable to the frailities (inability to remedy unappropriate actions) of all such devices.

1.2 Flow-Chart Representations A series of unconditional commands could be represented as a 'flow-chart'; it would consist in a connected series of operation or assignment boxes



and so on. Generally, and non-trivially, flow-charts are used to represent conditional processes and to admit of feedback control as well as feed-forward control. Consequently, the notation includes a condition test symbol (usually shown as '◇') and such 'test boxes' are combined freely with unconditional operation boxes. Some caution is needed in so far as the flow-chart may either represent a process (like an unmonitored serial computer program) in which all of the data is available within the system or it may represent interaction with other systems. For example, some operations may be searches or enquiries that are not guaranteed to terminate; some of the conditions tested may depend upon the answer to genuine questions (rather than look up operations within a fixed data structure). The crucial point is not, of course, where the operational data is located but whether it is available to the system (i.e. the operation is do-able). You

cannot drive to Manchester from the garage at your office starting at once (on the face of it a reasonable command), if there is no motor car in the garage at this moment, nor can you check stock prices from the telex, if the line is out of action. If the availability of motor cars, telex data, or whatever, is aleatory then the system receives an input. More realistically, 'if the best restaurant in town is full tonight, book him a table at his hotel; otherwise book a table for five at the best restaurant' generally does demand an enquiry (a non-trivial one; 'which is the best restaurant?' and 'is there such a thing?). So far as prescribing a process is concerned, the existence of genuine input as against the existence of guaranteed data-values is a matter of what is assumed (the *use* of a prescription or the manufacture of a *description* is a very different matter). Unless the contrary is stated, we shall assume that the objects used to perform operations (motor cars, slide-rules,

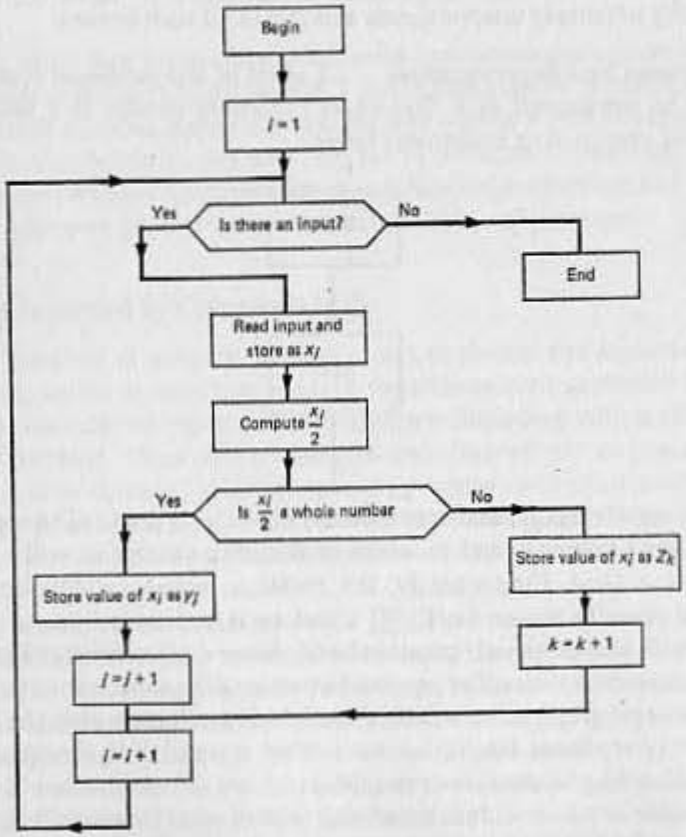


Figure 85 A performance algorithm flow-chart for the process of taking a list (x_i) of numbers and producing a list (y_j) of even numbers and a list (Z_k) of odd numbers.

aeroplanes) are available, that there are no genuine inputs, and that for any condition test the posited conditions exist, so that a value can be determined. In other words, we are making the temporary and very dubious assumptions (a) that the boundaries of a system are determined (whatever that means; it may mean many things depending upon how the 'system' is dissected out from the flux of events as '*the* process') and (b) the system is a *closed* system (of these (b) is meaningless unless (a) is affirmed and the affirmation or denial of (a) is actually a very subtle issue).

With these caveats in mind it is useful to distinguish several varieties of flow-charted process; performance algorithms, decision flow-charts (condition test charts) and subroutine charts.

A *performance algorithm* flow-chart (for example, Lewis, Horabin and Cane, 1967; Lewis, 1970) consists in conditional test boxes and operation boxes linked by directed arcs. It has a beginning and an end (Fig. 85). It may contain loops (iterated conditional operations). Execution of a task can be represented by placing *one* marker on the beginning and moving it along the directed arcs. On reaching an operation box, the stipulated operation is carried out. It may be any imperative (an assignment statement like 'give a variable a freshly computed value', or a physical action like 'place the form on a pile'). On reaching a test box, the test is carried out and an appropriate exit is made. Strictly, the tests are confined to evaluating variables that stand for predicates with the values 'true' and 'false' (or '1' and '0') and to tests for equality on numbers. But these tests may be combined in a single test box by the logical connectives 'and', 'not', 'or', in which case the test box is called a limited entry decision table. Further, the value □ (no value assigned) is admitted with respect to undetermined exits.

The addressee is credited with the ability to interpret and carry out all the instructions (both tests and operations) that appear in the algorithm.

A decision table flow-chart has the same connection rules and the same 'one *marker* at once' restriction as the algorithm. But the test boxes may be Decision Tables (Fig. 86) rather than limited entry decision tables. In these boxes, various conditions can be tested, for example, inequalities between numerically valued variables like 'greater than' or 'less than' or 'greater than or equal to' or 'more than 100 and less than 250'. These conditions indicate the holding or not holding of relations. Though the relations cited so far hold between two variables it is possible to extend the notation in order to encompass many-variable relations such as 'A=B=C=D=E=F' ('the books are balanced', or 'the aircraft is symmetrically loaded'). Once again the addressee is credited with the ability to interpret and carry out all of the instructions.

Several difficulties arise in connection with the extended form of decision chart. Under the broader interpretation of a condition it is not clear what

Determination of Tax Liability	*Column or 'Rule' Number*					
	1	2	3	4	5	6
Q1 Selling Price greater than Market Value?	Y	Y	Y	N	N	N
Q2 Market value greater than Cost price?	Y	N	N	Y	Y	N
Q3 Selling Price greater than Cost price?		Y	N	Y	N	
D1 Tax charged on Selling Price less the Cost Price, less Expenses	X					
D2 Tax charged on Selling Price less the Cost Price, less Expenses		X				
D3 No Tax either charged or allowed			X	X		
D4 Tax allowed on Cost Price less the Selling Price, plus Expenses					X	
D5 Tax allowed on Market Value less the Selling Price, plus Expenses						X

Figure 86 A decision logic table for determining tax liability (after Lewis, 1970).

can be compared with what. Whereas the addressee or user may be expected to compare *numbers*, does he have the ability to recognise more obtrusive properties; either abstract properties like *similarity* or concrete properties like shape and colour that are used in many ordering schemes?

The dilemma is usually resolved (or simply inched around) by appeal to 'well learned skills'. The solution is legitimate enough if training in these skills is provided before the user is asked to interpret the chart; failing that, he is almost bound to interpret any command to *do* (not just the command associated with the decision table) as a command to learn. If so, the interpretation of the entire chart becomes equivocal; various operations coalesce into 'more or less well learned skills' and the chart becomes virtually meaningless (it might as well be represented as a big decision table).

Suppose, for one reason or another, that a process *is* serially executed. It is thus possible to represent it as a flow-chart. The chart may not only be complex but repetitive. Like any other serial programme, it usually contains parts (i.e. subroutines) that represent the same kind of computation carried out in different contexts or with different data. A 'subroutine chart' is a representation of this structure. It has elements that may be either simple conditional commands or subroutines made up from many; the only

restriction is that each subroutine has been delineated, before execution, *in extenso* (as a segment of simple conditional commands, duly surrounded by a marker to indicate its integrity and named or labelled). Such a chart is shown in Fig. 87 and it can be employed practically using instruments for accessing and assigning parameters to representations of the subroutines. Plate 8 shows one of them.

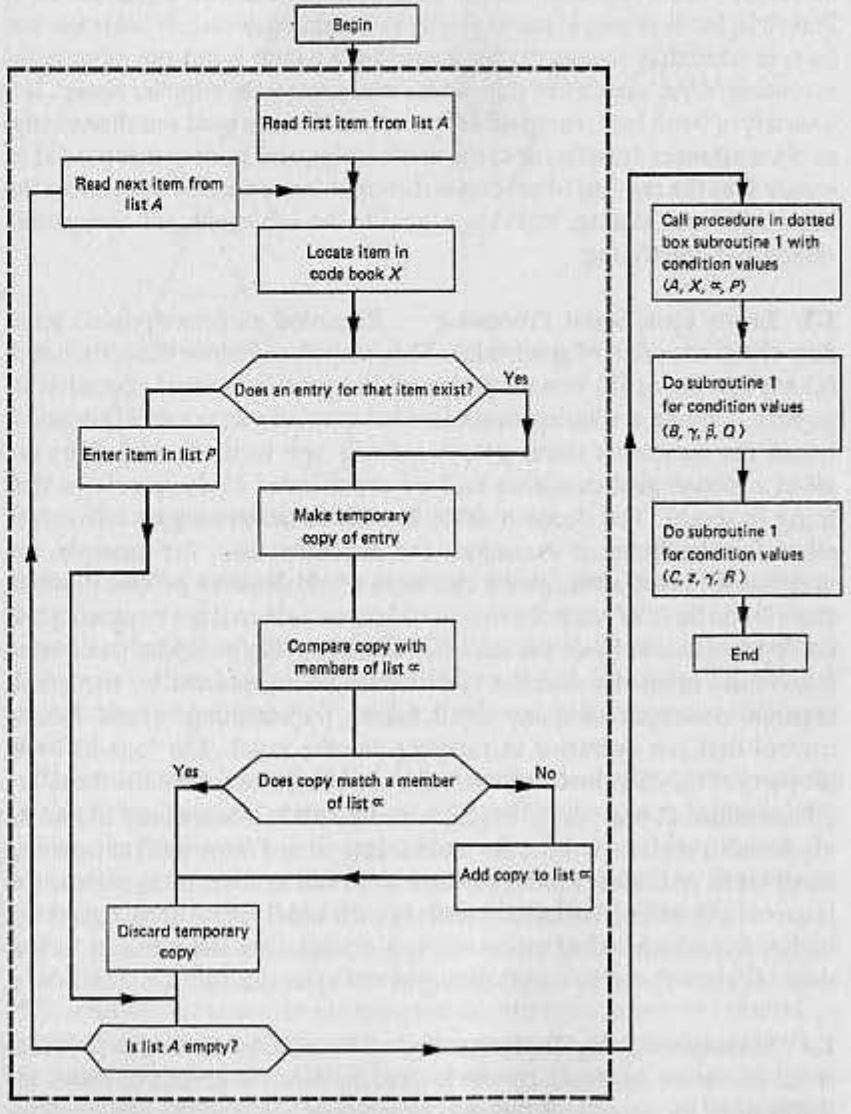


Figure 87 Example of a subroutine chart.

Man's habit of interpreting commands to *do* as commands to *learn* is contingent upon naming and describing what is to be done, i.e. upon a goal condition being specified. But the naming of a subroutine goes together with a contextual statement of what it achieves or satisfies. Consequently, subroutines (especially the simpler ones like index searches) become well learned skills. That is harmless and may be desirable provided the chart is no longer held to represent what is actually done. The instrument shown in Plate 8 is, for example, in use explicitly as a training device. What is learned (say, in searching documents for a supplier's name) is not the more rapid execution of the 'search for' subroutine with parameter 'supplier name'. It is a variety of (with luck, compatible) processes that are used simultaneously, as circumstances demand, or at the user's whim, and some care is needed to ensure that the rag-bag of processes (brought into existence by naming the subroutine and stating, thereby, a goal to be achieved), are compatible rather than interfering.

1.3 Limits upon Serial Processing Regarded as prescriptions, serial flow-charts are often of great value. They may also *describe* the activities of (a) any novice, or (b) some experienced but peculiarly serial operative. In general, however, the human brain does not normally act as a serial processor (recall the comments about people *learning* new methods when they are given a named goal condition and are commanded to *do* operations that bring it about). The decision table flow-chart constitutes an attempt to allow for simultaneous execution; the condition tests, for example, are unordered. Hence, although the chart as a whole depicts a process in which there is one locus of control at once (which can be simulated by moving one token from box to box), the activities actually going on whilst this (main) token rests upon the decision table would be represented by the simultaneous movement of many small tokens (representing several loci of control that are operative in carrying out the tests). The 'one at once' property of the chart itself is preserved by insisting that the 'small tokens' are all assembled at one point, 'begin', when the main token comes to rest on the decision table box. Also, the main token cannot move until at least one small token resides on a point labelled 'end', and as soon as the main token is moved away from the decision table box the small tokens are assigned to a limbo, from which they return to 'begin', and that returning to 'begin' depends upon the main token coming to rest upon the decision table box.

1.4 Permission-Giving Representation of Non-Serial Processes Non-serial processes are generally represented by directed graphs in which the nodes stand for conditions that may or may not hold and the arcs stand for relations of precedence between the conditions and their holdings. A con-

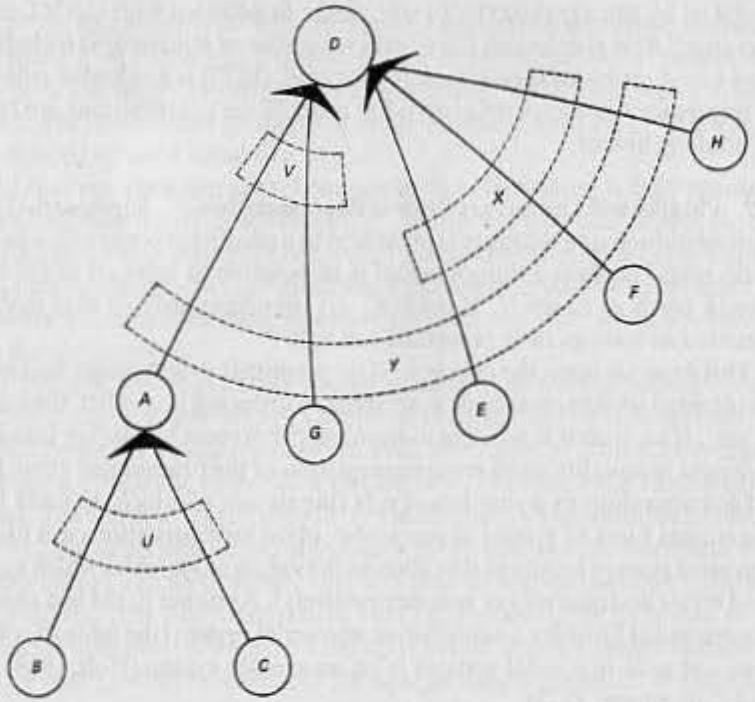


Figure 88 Fragment of a precedence network.

dition is itself a relation (not a state). A typical part of such a graph or *precedence network* is shown in Fig. 88 where there are two notations; condition *A* may hold if and only if conditions *B* and *C* both hold; condition *D* may hold if and only if either conditions *E* and *F* and *H* all hold, or if conditions *A* and *G* both hold, or if both of these complex preconditions are satisfied.

The precedence network is clearly permissive as all of the transitions are prefaced by 'may'. The operations that bring about condition-holding (which may be executed if appropriate commands are issued and the command preconditions hold) are usually represented as labels over clusters of arcs in the graphs (for example *U*, *V*, *X*, *Y*, in Fig. 88).

Pictures of a similar kind are familiar enough. For example, they are used in critical path analysis (for representing production processes) and in PERT networks (where the durations of a critical path chart are refined into expected durations and other statistical parameters, such as the variance or the means of expected value). In fact, these are all rather specialised forms of process representation (so is Fig. 88 which would be known in computational circles as an and/or network). The whole subject is comprehensively

reviewed by Elmagrahby (1970) who deals, in addition with GANT charts and signal flow graphs and introduces a number of innovations including a very useful *restricted conditional*. Hobbs *et al.* (1970) is a valuable collation of papers for the computer expert, but most of the contributions are rather technology bound.

1.5 Parallel and Concurrent Process Representation Suppose that such a permission-giving structure is presented to a parallel processor (the human brain might be such a thing), would it be possible to issue all of the commands (such as to do U, V, and X, Y,) simultaneously so that they are executed as soon as their preconditions hold?

This depends upon the network. If no permitted subprocesses that might be executed simultaneously or in an arbitary order are in conflict, the answer is 'yes'. If so, then it is possible to simulate the process by placing tokens on sufficient preconditions to ensure completion of the process and move these tokens according to a simulation rule (the nature of which depends upon the chosen form of graph; in particular, upon how conditions are used to represent storage locations that contain the values of variables which may be read either destructively or non-destructively). As noted in the last chapter, the canonical form for a conflict-free non-serial system (the issue of conflict does not arise in a serial system) is an occurrence system (Holt, 1968; Holt and Commoner, 1972).

There is also a sense in which viable systems must be closed, i.e. that 'alien tokens' do not intrude, except as transients, into a simulation. The tacit suggestion is that 'a process' or 'a system' is identified by the token type (colour, perhaps) used to simulate its execution. For a process with input, some preconditions are determined by 'other' processes with different types of token and the systems must not get mixed up. Though there is a kernel of truth in this notion it is replaced, in the next volume, by a much less artificial identification of a process's individuality based upon essentially immunological criteria abstracted from biology. The kind of 'other recognition' (hence, in a limited way, 'self recognition', which is a ubiquitous characteristic of any biological immune response process) appears to be essential to the integrity of all processes and systems (it is the key to what is meant by 'begin' and 'end' as well).

1.6 Resolution Suppose there is conflict. How would it arise?

(a) It may be due to any process that looses specificity or makes data ambiguous (the same thing). If the computation is fully specified in advance, this kind of conflict may be avoided by a precedence network that prohibits it. For example, in computing the value of

$$(a + b) \times (c + d)$$

the '+' of (a,b) and the '+' of (c,d) may be computed in any order, provided the results $e = a + b$ and $f = c + d$ are stored for later reference (if the storage is available) or the '+' operations might be carried out simultaneously. But both values (e and f) must be available before the '$\times$' of (e,f) is computed to yield value $= e \times f$.

But it is not always true that the required computation is fully specified (in the requisite sense) in advance (e.g. any open-ended search process).

(b) Conflict may also arise because of a command to perform incompatible physical operations; for example, you cannot brake and accelerate at the same moment (assuming you have two feet and one of them is resting over the clutch pedal).

(c) Perhaps the commonest source of conflict is storage allocation (as mooted in the example of (a)). Computer users are apt to take storage for granted (machine designers have no such illusions). If man carries out a computing operation, typically a clerical one, then the issue of storage is crucial. The storage is a resource. Some storage is located in external media such as files, cards and so on. Some storage is located in his own brain and conflict occurs due to processes that can find no storage position for computed values, that overwrite values, that fail to create free locations or to extend the (external) storage to a requisite extent before a task is started.

(d) Finally, conflict can occur because of communication. Though the fact was not stressed, all of the examples so far cited are due to unwanted communication between loci of control-executing (incompatible) operations. If these loci are conceived as different people (in a team, for instance) there is no difficulty in seeing the point. But the loci of control may just as well coexist in the same one brain processor.

The situation is greatly complicated as soon as there is a genuine interaction between *our* process and some other process; whenever *we* are in genuine ignorance about what the other process will do. For example, the sanitary prohibitions of (a) cannot be realised.

1.7 Communication and Cooperation Communication is a mixed blessing and the last couple of paragraphs emphasised its negative aspect. On the other side of the coin, communication between loci of control (they could be in separate people but need not be) gives rise to cooperative effects which enable otherwise impossible computations. Similarly, a mix of internal and external communication is responsible for the 'redundancy of potential command' of McCulloch (1965). Of various potentially dominant loci of control, the one that is currently best informed about the prevailing situation becomes immediately dominant. In all cases, the communication depends upon a partial or complete synchronisation between the loci of control.

In a very fundamental sense, this synchronisation is information transfer, and the communication is about that information transfer (or is a symptom of it). For later reference I have shown in Fig. 89 Holt's most elegant example of the distinction between 'information transfer' in this sense (far removed from selective information theory) and 'control'. The caption is self explanatory.

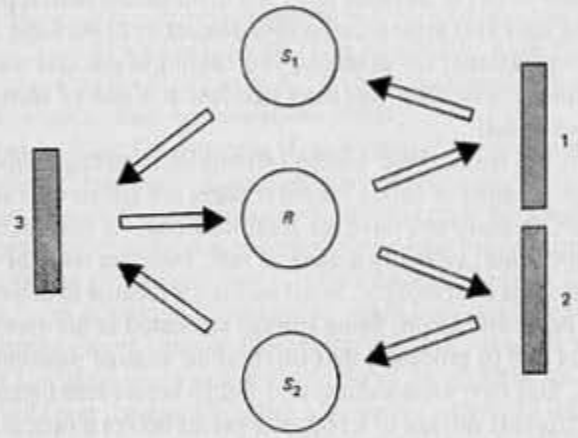


Figure 89 Simplest case of a Petri net showing conflict (after Holt *in* Bateson, 1972): circles, system conditions; hatched bars, transitions. Starting at condition *R*, two different transitions might occur (1 or 2). A determination of which transition occurs requires *information transfer* from *outside* the system.

1.8 Some Comments on Human Processing The possibility of conflict occurs with the possibility of cooperation; competition is a special case of conflict. Processes that entail all of these possibilities are concurrent (partial synchronisation of otherwise asynchronous loci of control is permitted).

It looks as though the human brain, taken as a processor, computes concurrently; that is, even if presented with serial commands to *do* these are interpreted at the least provocation (perhaps always) as commands to *learn*, and, in fact, how to learn how to perform the same task as executing a process.

As noted in the last chapter, the canonical representation for concurrent systems is a Petri net. The difficulty is that token assignment simulation rules are hard to formulate for a general case, though in special cases there is no great difficulty. On the other hand, for special cases that are tractable, it is often just as convenient to use the simpler notation of Fig. 88 in a slightly augmented form. If a human process-prescription is to have descriptive pretentions (or even wide utility as a prescription) then a

representation of this type is mandatory as a minimum. An example is shown in Plate 9. The networks are complex because human processes are complex. The representations are the least able to furnish a workable approximation to reality.

2 Tutorial Programmes

Training routines are typically organised as temporal sequences or programmes which often contain conditional operations and test points. The teaching of academic subject matter is sometimes represented in the same manner. The executive agent may be a teacher, a student himself, or a machine; the last case is emphasised because the limitations are more clearly expressed, not because machine administration is 'best' or even 'most frequently employed'. Unlike task performance specifications, the instructions in the tutorial programme are intended to bring about changes in a pupil or student. They may, incidentally, evoke performances (either exercises or tests) but they are not so much directives to *do* as directives to *learn*. Tutorial programmes are used as prescriptions and (locally only) as permission-giving structures that enable certain autonomous activities and disallow others. They are not used to *describe* learning though any defensible programme is based upon a descriptive (and, if possible predictive) model for how students do learn. This model justifies the programme design.

The minimal capabilities of the executive agent depend, first of all, upon the programme type.

The simplest (serial and *linear*) tutorial programmes are simply chains of instructions. Logically they have the calibre of *game-playing strategies*. Whoever designed the programme had in mind a 'game' in which his executive agent was one player and a member of the student population the other player. Before play, the designer set out all possible joint moves leading to an outcome and selected one sequence as the playing strategy to be adopted by his executive agent (which need thus be no more than a counting shift-register moved from one address to the next either by an internal clock-pulse or a 'self pacing' clock-pulse received from the student). Alternatively, the executive agent, together with the tutorial programme, may be envisaged as a simple feedforward of predictive control mechanism.

The next kind of tutorial programme (a so called 'branching' programme) is a *Banerji game playing strategy*: that is, it contains conditional tests and conditional assignment or execution statements. It can be represented by an algorithmic flow-chart and thus as the specification of a finite automaton. The designer is able to stipulate contingencies (what is to be done *if* such and such occurs) and the minimal executive agent is more sophisticated; a finite-state machine of whatever complexity is required. Alternatively, the

executive agent, together with the tutorial programme, may be envisaged as a feedback controller (perhaps with feedforward or predictive subsystems).

The 'liberalised' algorithm or condition chart calls for a still more elaborate executive agent since the order of operations is no longer always determined (one class of examples is cited; namely, the structural communication method). It is quite possible to conceive tutorial programmes that correspond to Petri nets and execution graphs as well, but none of them are called 'tutorial programmes' (these structures are characteristic of conversational teaching systems, for example, as discussed later in the book).

Form apart, tutorial programmes differ in respect of the operations they are intended to evoke; that is, there are differences in the intended interpretation. For example, from a *strictly* behaviouristic point of view a human student is regarded as a *target* organism. The prescriptive parts of a tutorial programme are seen as stimuli or combinations of stimuli that *cause* responses and the permissive parts as constraint relaxations that *enable* autonomous events which are otherwise prohibited. In contrast, if human students are seen, even in the limited context of instruction, as sentient beings able to interpret commands, questions, and the like, then the prescriptive instructions either figure as commands to learn (for exercise material and tests, local commands to *do*) or questions that elicit answers. The permissive instructions remove some of the sanctions imposed upon the student by a tutorial contract in which he agrees to interpret and obey the tutorial progamme. The major difficulty is that though *explanations* may be elicited by *questions*, an explanatory reply cannot usually be evaluated except by the student; in contrast the answer to a multiple-choice or a which question can be evaluated by quite a simple executive agent. As a result, nearly all the conditional tests involve degenerate evaluation (done by the executive agent) and the tutorial efficacy of the scheme is thereby restricted.

It is worth pursuing this point if only to justify an otherwise pernickety insistence that an algorithm *is* an algorithm (or a tutorial programme *is* a tutorial programme) only if its interpretation can be properly specified. The fact that explanatory replies cannot (in general) be interpreted, brings out an important limitation which is obfuscated if the tutorial programme is conceived naively as a 'plan for making things happen'.

None of the tutorial programmes we consider are adaptive with respect to an individual student and only one of them has an explicit mechanism for adaptation with respect to a student population.

2.1 Tutorial Linear Programmes A serial and 'linear' teaching programme can be administered by programmed texts, flash cards, or the like.

Suppose the programmed materials are presented to the student by a simple machine. The sequence of operations carried out by the machine is as follows.

1. Some descriptive or explanatory material is presented in writing, pictorially, as 'frame' or 'item' number, n, in a linear sequence.
2. A question is asked about the material in frame n. This poses a problem.
3. The student tries to solve the problem or respond to the question either (i) by constructing a written response (filling in within a blank on the frame itself) or (ii) by selecting one of the several multiple-choice alternatives; for example, by pressing a button corresponding to the chosen alternative. Generally, the student is allowed to respond at his own pace, but occasionally a mild form of time constraint is imposed to prevent dawdling or downright inattention to the material.
4. The student indicates that he has completed his response (pulling a lever, pressing a button).
5. The machine presents the correct response to frame n so that the student can compare his actual response with the correct response.
6. The student is asked to contemplate the difference, if any, between the actual and the correct response and to remedy any misconceptions that led to this deviation.
7. The student indicates (for example, by pressing a button) that his comparison is complete and that he has taken the necessary steps to put his mental house in order.
8. The machine moves frame n, the nth response, and the correct nth response from view and exposes frame $n + 1$ in the programme.

Machines of this sort and the techniques that go with them have been used and developed by Pressey (1960), Skinner (1968), Holland and Skinner (1961), Glaser (1962), Markle (1960), Gilbert (1962) and many others. These operations do not dictate a programming principle any more than the pages and paragraphs of a book dictate its content and style. Nevertheless *some* programming principle *must* be adopted. Historically, most linear tutorial programmes are based upon the idea that behaviour can be shaped according to the principals of operant conditioning and upon the side assumption that knowledge or skill can be built up sequentially, in steps corresponding, roughly, to the frames. Since the programme is linear (frame n is always followed by frame $n + 1$) there is also a supposition that one good sequence of presentation of the material can be chosen for all individuals in a target population.

My own extension and criticism of this view of learning is given, at some length in Chapter 11 and the following account also provides background data for what is being modelled, criticised, and (in part) rejected.

The precepts of operant conditioning are that operant responses are established by contingent reinforcement and that a successful response is inherently reinforcing. Hence, linear programmes are generally written to secure an expected correct response percentage in the order of 80 to 90 per cent. Apart from the establishment of correct operants, there are four main processes involved in the underlying learning model.

(1) Chaining of responses into sequences wherein the response to an item evokes the discriminating stimulus eliciting the next response (the discriminating stimulus signals an occasion on which the response may be reinforced. It thus also serves as a conditioned reinforcing event). (2) The development of 'higher level' reinforcers. (3) Discrimination, whereby responses become more selective. Finally, (4) 'generalisation', whereby responses are extended to classes of similar stimuli.

Using a 'classical' technique, the programmer specifies a terminal behaviour he would like the student to exhibit (commonly he designs a post-test to determine whether or not this terminal behaviour is achieved) and writes a sequence of frames $1, \ldots, n, n+1, \ldots, N$ to shape the student's current behaviour so that the terminal behaviour is approximated. The subject matter is thus broken down into segments and the size of the segment that goes into a frame depends upon the expected value of the student's operant-span, i.e. the size of gap he is able to fill in answering one of the questions whilst having an 80 or 90 per cent probability of being right. Some of the frames, interpolated at suitable points in the sequence, are intended to instrument chaining operations, other to carry on discriminations, and others to induce generalisations. To secure these ends, the programmer has a repertoire of standard tricks at his disposal which refer to the minutiae of the conditioning process. Perhaps the most important are (a) the provision of cueing or 'prompting' information which partially specifies the correct response, and (b) the converse operation of stimulus 'fading'. By the provision or withdrawal of cues, it is possible to adjust the correct response-probability with respect to a given type of material. Of course, the self-paced presentation allows the individual student to adjust his load within quite wide limits. He can go rapidly through the frames he finds simple, and slowly through the difficult ones.

If behaviour-shaping is taken literally, then contingent reinforcement means associating a behaviour that normally occurs autonomously and with high probability, that is, 'because the target organism likes to behave in that way' with an initially low probability behaviour that is to be established. The protoganists of behaviour-shaping are alive to the underlying difficulty; no one knows what behaviours are reinforcing and *a priori* probable.

In the past, the dilemma was resolved (or simply pushed aside) by appeal

to physiology (certain brain centres, if stimulated *do* produce pleasure and are stimulated by high probability autonomous behaviours). Whilst true, this kind of argument (which is, incidentally, part of a physiological meta-theory, not part of behaviourism itself) fails to account for obvious individual differences due, at least, to previous and uncontrolled conditioning of the target organism. Various schemes have been used to solve the joint problem posed by ignorance of what *is* reinforcing and the existence of individual differences due to esoteric experience. All of them go beyond the strict behaviouristic framework, though this point is often glossed. The most lucid scheme is the 'reinforcement menu' of Homme (1966). Students exit, periodically, from the programmed or scheduled activity, and (if they are to be reinforced) choose from a 'menu' of possible rewards before returning to work. The menu, however, is chosen and updated according to criteria that lie outside the learning model and are the responsibility of a 'reinforcement manager' (for example, the teacher in charge of an infant school).

The same procedures are justifiable, for very limited types of material, with respect to a systemic model in which teaching is assumed to establish goal-directed or problem-solving systems in the student's mind. At first sight, only the jargon is modified. The corrective signal (comparison of the actual response and the correct response) provides 'knowledge of results' information rather than reinforcement and the external goal-achievement test implied by the provision of the corrective knowledge of results becomes internalised as learning proceeds. Cueing information partially solves the problems posed by the stimuli and may reduce the overall goal to subgoals, but some care is needed (Chapters 5 and 8). Beyond very restricted situations and learning conditions this reinterpretation entails much *more* than a change of jargon and notation scheme.

The 'classical' technique is supplemented by a more systematic behavioural method called mathetics, chiefly developed by Gilbert (1962) and his associates. As before, a terminal behaviour is stated, but this is systematically transformed into a 'synthetic prescription' that states how the student's current behaviour is to be modified in order to attain the terminal criteria. The basic units of the programme are no longer frames but 'exercises', and the small step-concept is de-emphasised or even discarded. The exercise brings about a specific behavioural change, and it entails instructing, cueing and observing stimuli over and above the discriminating stimulus that elicits the main or 'mastery' response. In other words, each exercise is a pattern of behaviour-changing operations; typically, it involves a sequence of frames in which a mastery response is demonstrated, prompted (or cued) and released. Further, in the construction of an exercise, the programme writer takes systematic account of the symbolic representation of the subject matter (the theory behind the behaviour). Mathetics is the best developed

and most elegant programming technique, but I believe its nomenclature is positively misleading. An 'attention-directing stimulus', for example, acts within the theory, as a command (not as a stimulus); a 'questioning stimulus' *is* a real *question* (again not a stimulus).

Whichever model is adopted, it is evident that the serial programme is a feedforward control process. The programme instructions control learning in so far as they are based upon a predictive model for how the student will react to them. The onus for the intimate control of learning is placed firmly on the subject's shoulders. (This comment applies very strongly to the mathetics programmes.) The student is responsible for pacing the trials and he acts both as comparator and corrective agent. True, a feedback signal is received by way of knowledge of results information, but it is used, in comparison and correction, by the *student*, and not by the *machine*. (The programme sets up conditions for internal, student-mediated feedback, but it does not itself exert feedback control over the learning process.)

It will be clear that not every sequence of frames constitutes a programme. To qualify for this title, it must be possible for the programmer to say what each frame is intended to do, and why (with reference to the criterial or terminal behaviour) it has been introduced at a given point (i.e. frames should establish the conditioned response '*x*' or 'make the discrimination *y*' or whatever). So far as behaviour shaping is concerned, only the *programmer* is required to exercise this much perspicacity. The pupil, subject or student is regarded (theoretically speaking) as a malleable entity on which the programme acts; albeit, an entity able to emit the operant responses that allow it to act. The linear format does not, of course, dictate a behaviour shaping view of the student (or vice versa). It is however well suited to this view.

2.2 Tutorial Branching Programmes A branching programme (again, imagine it is administered by a teaching machine) is an instrument capable of mediating feedback control over the learning process. Because of this the learning model employed in constructing a branching programme can be much less specific than the model for a linear programme addressed to the same subject matter and student population. Since it *is* feedback regulated, unknowns can be filled in during the execution of the programme; for example, the programme may prescribe a test for a condition and certain subsequent operations that are to be performed *if* certain test values are in fact obtained. By way of contrast, the model for a feedforward (linear) programme must, if it is to exert the same modicum of control, incorporate estimates of or assumptions about the values that will be obtained. The psychological background is correspondingly eclectic (behaviourism, cognitive psychology, motivational psychology, and so on).

The distinction between linear and branching programmes is illustrated in Fig. 90, where (*a*) represents the arrangement of frames (nodes, in the network) for a linear programme and (*b*), (*c*), (*d*), (*e*) correspond to various types of branching sequences.

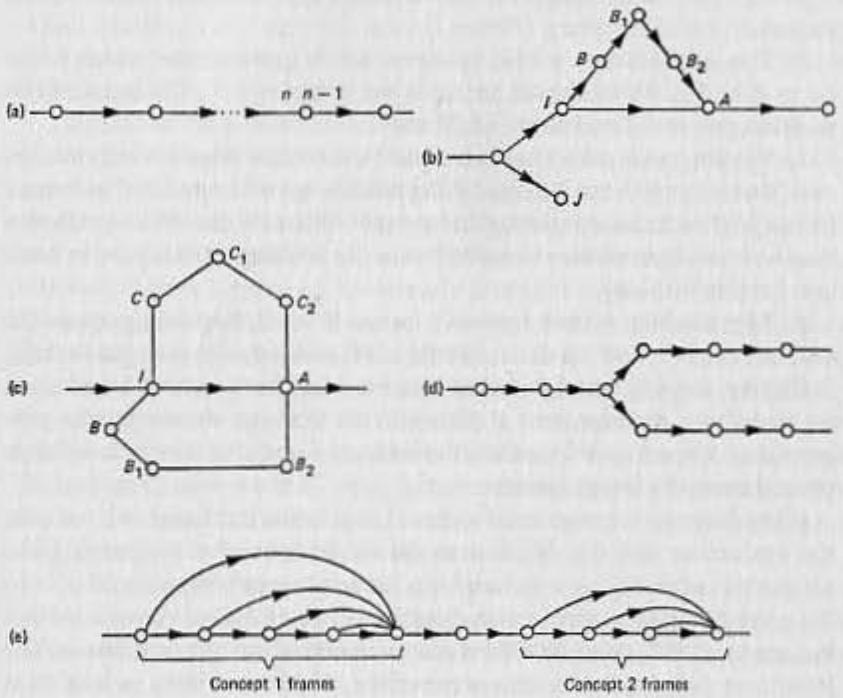


Figure 90 Forms of programme.

Suppose, for example, the student finds himself, on trial *n* in a teaching process, at frame *I* in Fig. 90(*b*). (It will be evident from the figure that being located at frame *I* on trial *n* depends upon a previous path through the network, i.e. the student could have reached frame *J* on trial *n*.) The following operations are performed by the teaching machine or the student.

1. As in a linear programme, the frame *I* material is presented to the student.
2. A question is asked about frame *I*.
3. The student is required to respond either constructively or selectively.
4. The response is evaluated. If the response is selective and mechanised (for example, button selection) then it is evaluated by the machine relative to some criterion. In the least elaborate case the evaluation consists in a comparison between the response made and a set of responses which, the

programmer has predicted, might have been made. This set certainly contains the correct response. But it may also contain response possibilities which, if they are selected, indicate the existence of misconceptions about the material in frame *I*.

5. As in the linear case, the machine presents the student with the correct response to trial number *n* (Frame *I*).

6. The machine uses a built-in decision rule to determine which frame (*A* or *B* in Fig. 90(*b*)) should be displayed at trial $n + 1$. The input of this decision rule is the evaluation of (4) above.

7. The student is asked to contemplate the correct response information and (if necessary) to use it in modifying his concept of the material in frame *I* (trial *n*). When he has considered the matter sufficiently, the student indicates that he is ready to receive the next frame (by pressing a button, or in some comparable fashion).

8. The machine selects the next frame *A* or *B*, depending upon the evaluation of (5) and the decision rule of (6) and presents it to the student.

Clearly, steps 4, 6 and 8 mediate external feedback-control based upon an evaluation function and a decision rule that are chosen by the programmer. Steps 5 and 6 mediate the internal corrective feedback which is present even in a linear system.

If the response is constructed rather than selected it is usual to hand over the evaluation and the decision to the student, i.e. the student is given certain rules (saying how to weigh up his constructed response relative to the correct response) and certain directives (if you feel that your evaluation is *a*, go to *A*, if *b*, go to *B*). The student himself selects the next frame. The important point is that a subsystem of the student has been isolated as a supervisor and controller of his own education, using the rules and directives that are given. This may or may not be a good thing to do depending upon the student and the subject matter. The expedient certainly does impose a real cognitive burden.

Several different sorts of programme network may be appropriate according to the intention behind the decision rules. The case just considered in Fig. 90 (*b*) is a network structure containing a mainstream linear programme with remedial loops. In the simplest possible conditions the response to frame *i* is evaluated as right or wrong and a decision is made to select *A* (for presentation at trial $n + 1$) if the *n*th response (to frame *I*) is right, to select *B* if it is wrong. Frame *A* is part of the main stream of instruction; in a linear programme it would be the next item after frame *I*. Frame *B*, on the other hand, is the start of a remedial loop B, B_1, B_2 which contains frames intended to iron out the misconceptions about frame *i* manifest by the evaluation 'wrong'. The student is returned to the main instructional stream at frame *A* only when he has shown evidence that his

misconceptions are put right. If the evaluation and the decision rules are more elaborate, the remedial action can be more discriminating and the programme structure may, with advantage, be more detailed. For example, in Fig. 90(*c*) there are two different sorts of remedial loop (B, B_1, B_2 and C, C_1, C_2) one of which is selected when the subject is 'wrong' at frame *I*; *which* one is selected depends upon *how* he is wrong. The use of remedial loops is akin to a cueing procedure extended over several frames. Because it is extended, it can be a controlled cueing procedure.

The evaluation and decision process can also be used as a mental testing facility to determine what sort of individual each student is, and to prescribe essentially different forms of instruction for different sorts of individual. A typical programme network is shown in Fig. 90(*d*), where the first part is a test sequence for sorting students into classes and the remainder serve to administer several more or less linearly arranged courses contingent upon class membership. It is usual (as in this network) to provide retest facilities for reassigning a student from time to time.

In section 3 we present a tutorial control process founded on the skip linear-programme network of Fig. 90(*e*). It is a series of linear programmes in which each one is addressed to the inculcation of one concept. From time to time, a test of concept mastery is made and, if the concept has been mastered, the student is directed to skip over the remaining frames dealing with this concept. Hence the name 'skip linear'.

The first branching programmes were written by Crowder (1960) who also designed the first teaching machines to administer them. Many people have subsequently been active in this field: Galanter (1959), Lumsdaine (1963), and Stohlurow (1961), to mention only a few of them.

2.3 Structural Communication One special programming technique is the structural communication method of Hodgeson and Bennet (1967) and Hodgeson (1968). Any study unit (corresponding roughly to a programme segment) consists of the following sections:

1. A statement of the author's intention, that is, the underlying topic and goal.
2. A viewpoint which orients the student and leads him to explore the field.
3. A presentation of the background material relevant to the concept of the study unit.
4. An investigation in which the student is posed problems, say five of them.
5. A solution phase involving a response indicator. This is a page containing (say twenty) statements, all of which refer to some of the problems.

The student tackles each problem in turn and selects statements that he believes to be true.

6. A discussion guide. Contingent upon the student's selecting and not selecting certain combinations of statements in the response indicator; he is directed, through a number of stages of analysis, to study some of the 'discussion comments' which are described below.

7. 'Discussion comments' which remedy misconceptions, resolve ambiguities, make statements and encourage the student to justify his point of view. The comments also direct the student to do something by going back to the problems, the viewpoint, or the intention.

Progress through this system is logical and systematic enough. Superficially, the student adopts a serial path which is recordable. He is not held in the straitjacket of sequential presentation and, broadly speaking, the wiser he is the less restricted he will be. Whereas the majority of branching programmes are algorithmic in the sense that they correspond to serial flowcharts, a structural communication programme can only be represented by a condition chart. For example, step (5) above is a condition test in which the required evaluations can be carried out in any order (usually with interruption and stacking of data permitted) or even simultaneously. By the same token step (7) contains unordered execution or assignment statements. Whether or not the student uses the freedom permitted varies a great deal but he is certainly, and rightly, encouraged to do so.

2.4 Discovery Learning Structural communication is a procedural reification of a doctrine called 'discovery learning' or (since undiluted discovery appears to be a pretty haphazard activity) 'guided discovery' and is a compromise between free rein and the authoritarian approach. The basic idea is excellent. Moreover, it is empirically supportable. Learning takes place if and only if the student is impelled to discover a connection between already learned concepts or to construct a concept *de novo*.

As often stated, however, 'discovery learning' is a sloppily made concession that learning is chiefly a matter of cognitive processing rather than 'behaviour shaping'. A few advocates of the principle take the trouble to be more precise, mainly in operational terms describing what tutorial acts should be performed to induce discoveries, rather than in terms of the mechanism involved. Thus, the work of Bennet and Hodgeson (cited previously) is one outstanding exception to a generally cavalier treatment of what on earth 'discovery' is, and the work of Belbin (1969) is another.

From a cybernetic point of view it is hard to condone an assumption that 'discovery' is a property of the mind without delving into the systems involved. This acid criticism is spurred on by a real hazard. There is no doubt that the cognitive operations called 'discovery' are very important

and that they are much more difficult to comprehend than 'adaptation' (induced by behaviour shaping). The majority of writings on the subject of discovery are intended to ease the difficulty, so that discovery is fictionally comprehensible as a popular tagword. In the process, 'discovery' is used to name a scholastic tautology (a rose smells sweet because it has sweetness; a mind discovers because it has the discovery faculty) as a result of which the word falls into disrepute and the excellent concepts, which actually underpin the discovery doctrine, fall with it.

2.5 Behavioural Objectives During the last decade or so one movement (above all others perhaps) has had a salutory influence in leading people to think clearly about training and education. The underlying dogma is 'behavioural objectives' (Gagne, 1962). As a terse and insufficient precis, the dogma holds, as follows.

(a) That the terminal criteria of training and teaching can be expressed in terms of behaviours (often very complex ones; for example 'multiply numbers *A* to *B* using a slide rule' or 'multiply numbers *A* to *B* using a table of logarithms' or 'multiply numbers *A* to *C*'). If these behaviours are correctly performed a student 'knows' whatever (multiplication, for example) is required of him. The word 'knows' is placed in inverted commas because of a contention that the galaxy of behaviours is all that an instructor can know (without inverted commas) about the student. (b) The complex terminal behaviours can be broken into segments and the correct performance of these smaller pieces of behaviour, usually induced by behaviour shaping, is a sufficient entry criteria for a behaviour shaping operation that inculcates the terminal behaviour (obviously (a) and (b) can be iterated with respect to each behavioural segment until very small pieces of behaviour remain as primitives and, in principle, until there is no residue).

The whole gamut of methods and models (with the possible exception of structural communication) are popularly adumbrated under the rubric, 'behavioural objectives'. Training programmes are certainly devised on that assumption and often more or less work (though how effectively it is hard to tell). The plain fact is, however, *if* discovery is taken seriously then neither the contention of (a) 'terminal behaviours are all *we* can know' nor the contention of (b) 'it is possible to segment any skill' hold water. If discovery does mean a cognitive event then we *can* and *must* know, in order to support this view, how a student comprehends a subject matter, describes it, and explains it. True, the data may come from objective records of what the student does but they are not interpreted as behavioural records. For example, physically and operationally there is a great deal of difference between 'multiplying' and 'explaining multiplication' though it may be the case that the *explanation* is physically manifest as making a *model* for the

multiplication process. The latter activity is neither interpreted as a 'multiplication behaviour' (it is, of course, a behaviour of some kind), nor is it viewed by the student as an elaborate multiplying response.

By the same token but in respect of (b) there is every difference between behaving and knowing. Some confusion is engendered by the fact that both of them are species of *doing*; in behaviour, overt operations are *done*; in knowing, operations are done to concepts, in order to combine them or to construct them. With rare exceptions, it is not possible to express *what may be known* in the same way as *what may be done* (if discovery is taken in earnest, that is). Given genuine discovery, not all states of knowing can be induced by a concatenation of behaviour segments or (as stated under (b)) inferred from the correct performance of a complex terminal behaviour.

These comments are not intended to dismiss the doctrine of behavioural objectives out of hand. Within a certain compass the doctrine has great and well attested utility. But either the definition of 'objective', needs revision, or (better, for 'objective' is sound enough in its proper domain) the limits of the dogma need to be recognised.

3 A Restricted Tutorial Programming Scheme

The following scheme was used for several industrial training application and, in collaboration with Xavier Salazar Resines (1966), at the National University of Mexico, UNAM, for academic instruction[1]. It is a better scheme than most, but the chief reason for dwelling upon it is to exhibit the defects which became apparent when it was put into practice. These defects are liable to beset any scheme which is based, as this one is, upon the ideas discussed earlier in the chapter. The defects are especially obtrusive if the scheme is intended (as this one is) to give substance to the notion of discovery.

The basic unit of mentation is called a concept and this is imaged as a goal-directed or control process. That is, a *concept* is an organisation which forms an hypothesis, acts to satisfy the hypothesis, and tests for its confirmation or denial. It may thus also be interpreted as a 'problem solver'. As in the last chapter, concepts exist at certain levels of control (parts of a hierarchy of control, designated *Lev 0*, *Lev 1* and so on. The *Lev 0* processes act upon an environment; the *Lev 1* processes act to construct or modify concepts placed at *Lev 0*; the *Lev 2* processes (if they exist) act upon *Lev 1* processes.

1. The work was strongly influenced by Roger Diaz De Cossio and Juan Cassias, also of UNAM.

Given the identification 'problem solver', learning is a problem solving *of* problem solving, i.e. learning, which is a *Lev 1* process, either constructs concepts, or remedies a deficiency in a repertoire of *Lev 0* processes.

A tutorial programme operates in concert *with* the *Lev 1* processes in a student's repertoire or *upon* the *Lev 0* processes in his repertoire (on a par with his own *Lev 1* processes). Hence, if the student's hierarchy of problem solvers (alias *concepts*) are placed in register with the tutorial programme, this programme prescribes *Lev 1* processes.

A discovery is defined as the construction of any *Lev 0* concept (in fact, various kinds of discovery are postulated since various kinds of construction are distinguished). A guided discovery is a discovery involving the cooperative interaction of a *Lev 1* process prescribed (for example) by a tutorial programme.

In order to compose *concepts* (goal-directed, problem solving processes) into aggregates, they are represented in a common manner. The chosen idiom was the 'TOTE' or 'Test operate test exit' unit (Fig. 91) introduced by Miller, Galanter, and Pribram (1960) rather than my own 'control units' (Pask, 1968, 1970b) which were less familiar to the users. Such elements (really 'If. . . then. . . else' statements) may be concatenated into sequences and formed into an hierarchy (Fig. 91). Any of these composite structures is also a concept. The representation is extremely restrictive (a) because it is confined to serially executed procedures and (b) because it is tailored to fit a *post hoc* account of execution in which the structure is obscured. However, with the augmenting notation given in the captions to the figures, it suffices and is peculiarly well suited to the representation of the operations induced in an executive agent by a tutorial programme.

Manifestly a TOTE unit (since it is really an 'If . . . then . . . else' statement) must be embodied in data storage and executed by some processor. Two storage and execution loci are postulated (both of them functionally rather than psychologically distinguished), namely, the 'intermediate storage' and the 'long-term storage' of the brain. Intermediate storage is a computing mill and is the seat of interference phenomena, and the like; long-term storage is essentially indelible. The other category of data storage, usually distinguished in the literature, is short-term storage (in which, for example, lists of data are rehearsed); it will be evident that the embodiment of any TOTE unit (hence, in this image, any concept) incorporates and organises a short term storage operation.

The primary optimising criteria has to do with intermediate storage from which *concepts*, if shown to be useful or valid, can be transferred into long-term storage. The criterion is as follows. Intermediate storage has a finite data capacity and may be over taxed. Any concepts coexisting in intermediary storage are liable to interfere under execution and thus to be demolished.

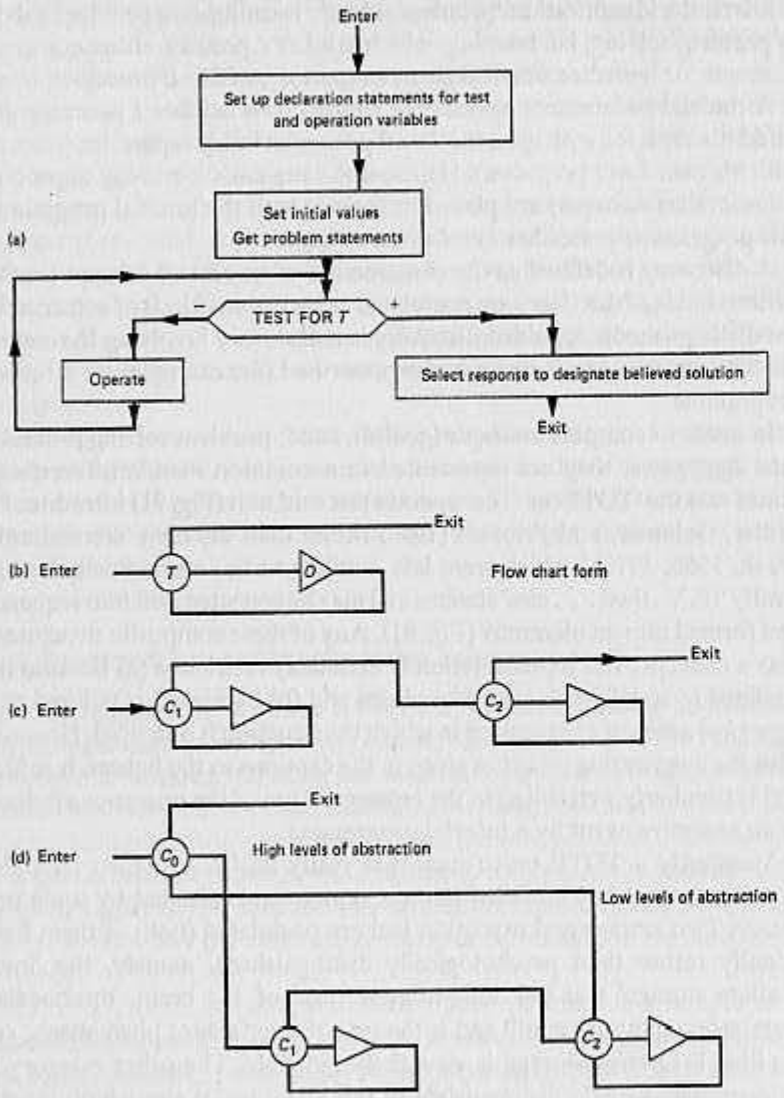


Figure 91 The TOTE (test, operate, test, exit, unit) notation and permitted compositions of TOTE units: (*a*) meaning of TOTE as an organisation; *T*, test for goal and *O*, operation performed upon information in immediate memory to approximate *C*; (*b*) shorthand form. All subsequent units are labelled in their test part either by *C* (the test for hypothesis under a concept) or *G*, *g* (educational goals and subgoals); (*c*) concatenation of TOTE units; (*d*) hierarchical structure of TOTE units. Concept C_0 is placed (very loosely) at a higher level of abstraction than C_1 and C_2.

Teaching 'sets up' a large number of *concepts* that are embodied in intermediary storage locations. If concepts are regarded in isolation any one of them is 'set up' by the cyclic tutorial operation shown in Fig. 92. A serial tutorial-programme thus loads intermediary storage with concepts; some of those established previously were needed as components in the next construction step. Of the previously established concepts, needed to form a particular concept at a given step, some may be retrieved from long-term storage where they are immutably inscribed but others have not yet entered long-term storage. These (and the concept being constructed) may be destroyed either by overload or interference and an effective tutorial programme should (by hypothesis) be designed to avoid either of these possibilities.

Tutorial operations are also conceived as TOTE units, set up by the tutorial programme, and intended to bring about the *tutorial goals* (*G*) of establishing a concept (*C*) for each topic in a subject matter instructed by the tutorial programme. A nested hierarchy of tutorial TOTE units is a *format*; in this case based on the *skip linear* scheme (the simplest branching network). This approach is founded upon two propositions.

1. Any long sequence (in principle, an indefinitely long sequence) of frames that are addressed to a single topic and are intended to inculcate a concept for this topic, may be regarded as a single TOTE unit (with the tutorial goal of teaching a concept for the topic in question). The validity of this proposition is a simple consequence of the fact that a tutorial cycle of the type discussed in section 2 *is* a 'test operate, test, exit' sequence provided that the data it operates with are interpretable by the student. As noted earlier, a tutorial TOTE unit exists at a level of control *Lev 1* and it *acts upon* concepts in the student's *Lev 0* repertoire though it also *interacts with* processes in the student's *Lev 1* repertoire.

2. Any subject matter has a main topic which can be broken down into subsidiary topics (a most dubious proposal).

However, if (2) holds, then, within the constraints of the *format* a tutorial programme consists in a prescription for an hierarchy of tutorial TOTE units that inculcate concepts for subsidiary topics (as tutorial subgoals), and that ultimately teach a concept for the main topic.

3.1 Construction Procedure 1. The main topic to be taught is given by the subject matter.

2. The main topic is reduced to constituent topics in the following manner (which closely parallels the arguments of Gagne, 1962, 1969). First observe that the main goal (to establish a concept of the main topic) could be achieved if (i) the student has a number of subsidiary concepts $C_1, \ldots, C_r$ and (ii) if he is provided with an appropriate sequence of performance

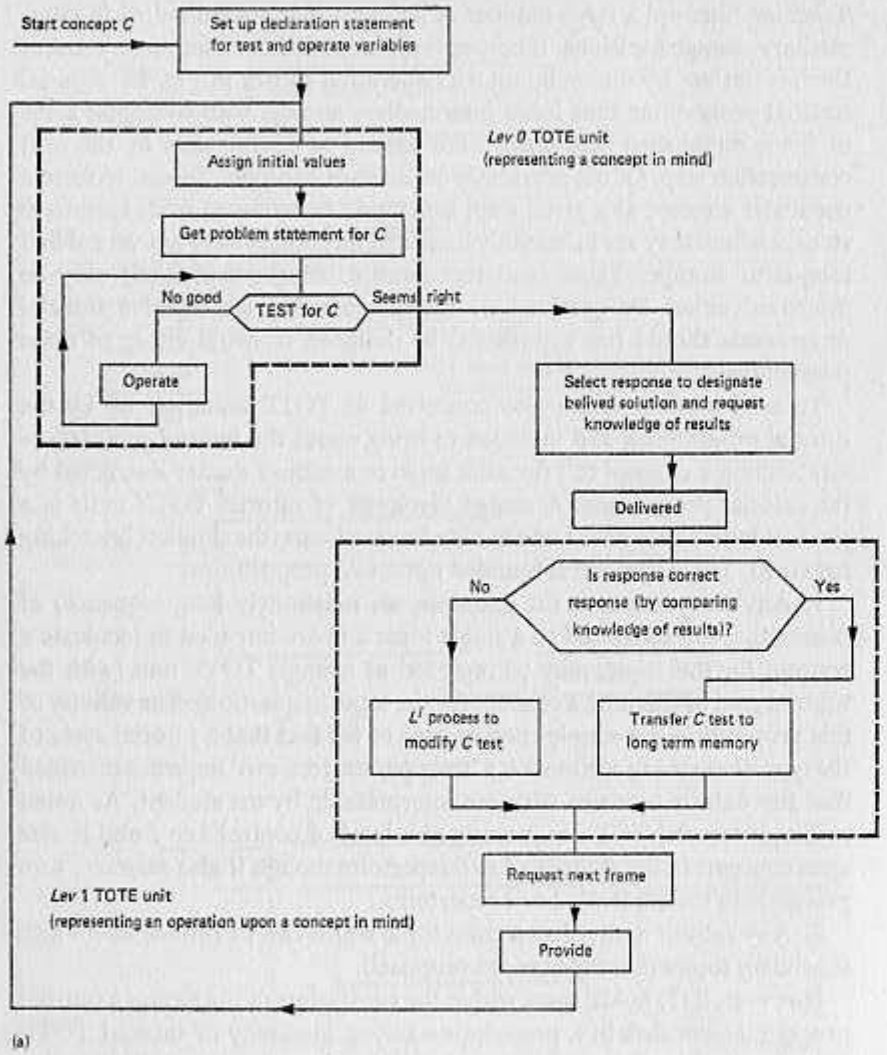
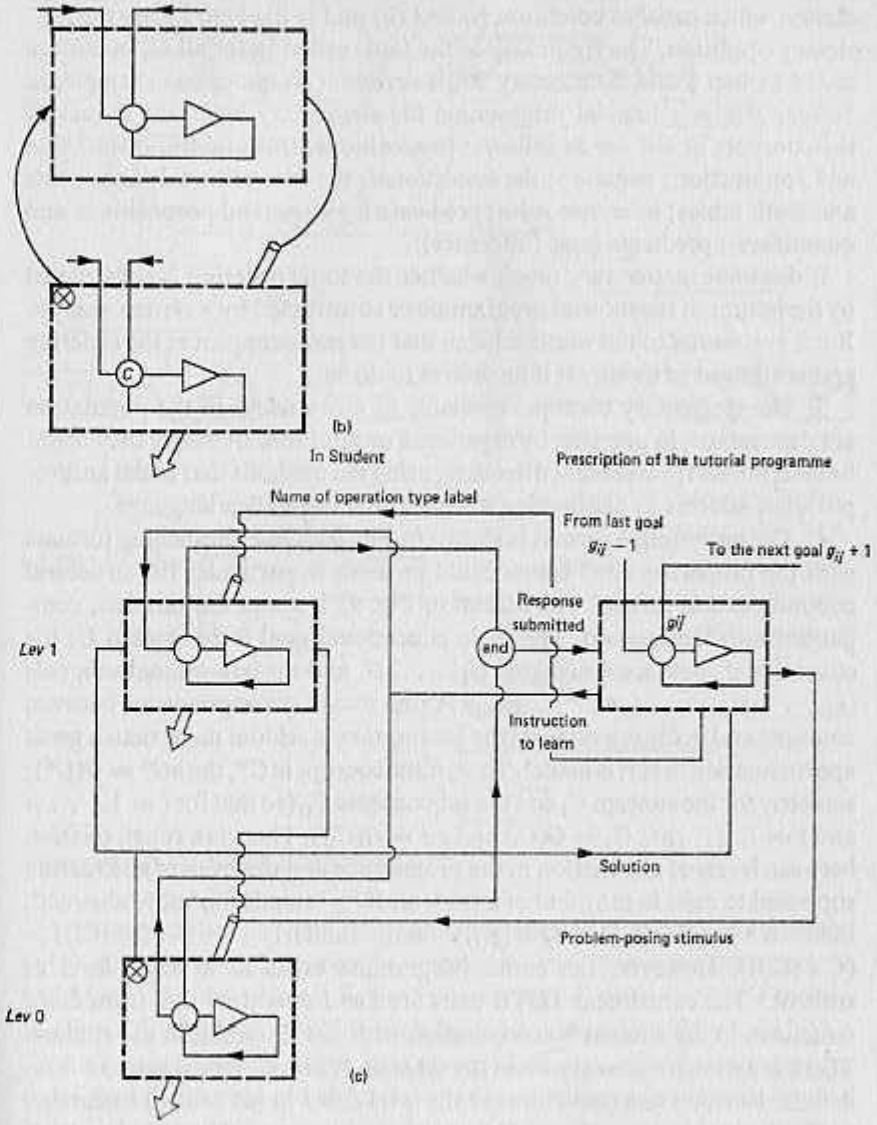


Figure 92 Representation of a tutorial cycle: (*a*) notation and meaning for any process; (*b*) shorthand form. A parametric arrow penetrating any dotted line indicates concept learning on student; an arrowed box delineates a TOTE unit which changes the data on which a TOTE unit tests or operates; a description specifies the TOTE units enclosed in the dotted-line box. (*c*) Relation between a tutorial TOTE unit (with goal G) prescribed as part of a tutorial programme and a TOTE unit (representing a concept C) in a student's mind. Each dotted box within the student may contain any composition of units which, for brevity, are represented as one unit in this chart.



instructions; concisely, a suitable algorithm that refers to the C_k; $k = 1, \ldots, r$. The problem is to select a breakdown of the main concept $BR = C_1, \ldots, C_r$ for which an algorithm is available. In general, this breakdown is not unique. There is a set, BR^* of breakdowns BR, from which one must be chosen. To narrow the choice, a further condition is introduced; namely, that for any BR in BR^* all of the C_k in BR must be associated with a legitimate concept ordering in the sense of step 6 (below). In practice some BR is

chosen which satisfies condition (i) and (ii) and is likely to satisfy the legitimacy condition. The legitimacy of the topic ordering for all C_k in BR are tested at step 6 and if necessary BR is revised. (To quote one example, in Salazar Resine's tutorial programme for elementary logic, the names of the concepts in BR are as follows: propositions, truth tables; disjunction and conjunction; negation; the conditional; the biconditional; arguments and truth tables; inference rules; predicate logic (general propositions and quantifiers); predicate logic (inference)).

It does not matter very much whether the topic ordering is constructed by the author of the tutorial programme or constructed by a system analyst. But it is *essential* to the whole scheme that the *author* approves the ordering and is allowed to modify it if he desires to do so.

3. The elementary concepts available to any student in the population are determined, in practice, by experience or intuition. In theory they might be determined from student discourse, using the methods that social anthropologists address to discovering salient words in an alien language.

4. The programme format is shown in Fig. 93. Other branching formats with the properties cited below could be used, in particular the structural communication format. The format of Fig. 93 is about the simplest, compatible with the method. The main educational goal is designated G; the educational goals are designated $G_1, \ldots, G_n$ and the educational subgoals $(g_{i_1}, \ldots, gi_{m_i}) \ldots (gn_{j_1}, \ldots, gn_{m_j})$. A one to one correspondence between concepts and goals is assumed (the assumption is seldom more than a gross approximation, but it is made). Thus, if the concept is C^*, the $nG^* = G(C^*)$; similarly for the concept C_i and the subconcepts C_{ij} (so that for $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$, $G_i = G(C_i)$ and $gij = G(Cij)$. There is a rough relation between levels of abstraction in the programme and the levels of abstraction supposed to exist in the mind of a student. If '>' stands for 'more abstract', then $G^* > (G \in \{G_i\}) > (g \in \{gij\})$ and similarly $C^* > (C \in \{Ci\}) > (C \in \{Cij\})$. However, the entire programme exists in a single level of control.[2] The constituent TOTE units are *Lev 1* operators that form *Lev 0* structures in the student by cooperation with *Lev 1* systems in the student. There is an inherent weakness in the scheme. Whereas the student can form definite learning sets (structures at the level *Lev 1* in his control hierarchy) the programme cannot do so. If it could, then there would be time variable relationships between the G_i and between gij (not merely an ordering $1, \ldots, n$ or $i_{j_1}, \ldots, i_{m_i}$). Further any programme with this property would be real-time and on-line adaptive; the adaptive process being formalised as a *Lev 2* box in the teaching system. The adaptive process of steps 8, 9 and 10 (below) is isomorphic with such a *Lev 2* box. But it does not act at the individual level, i.e. the programme is adapted only to a population of students.

2. Apart from cueing operators which may be regarded as *Lev 0* objects.

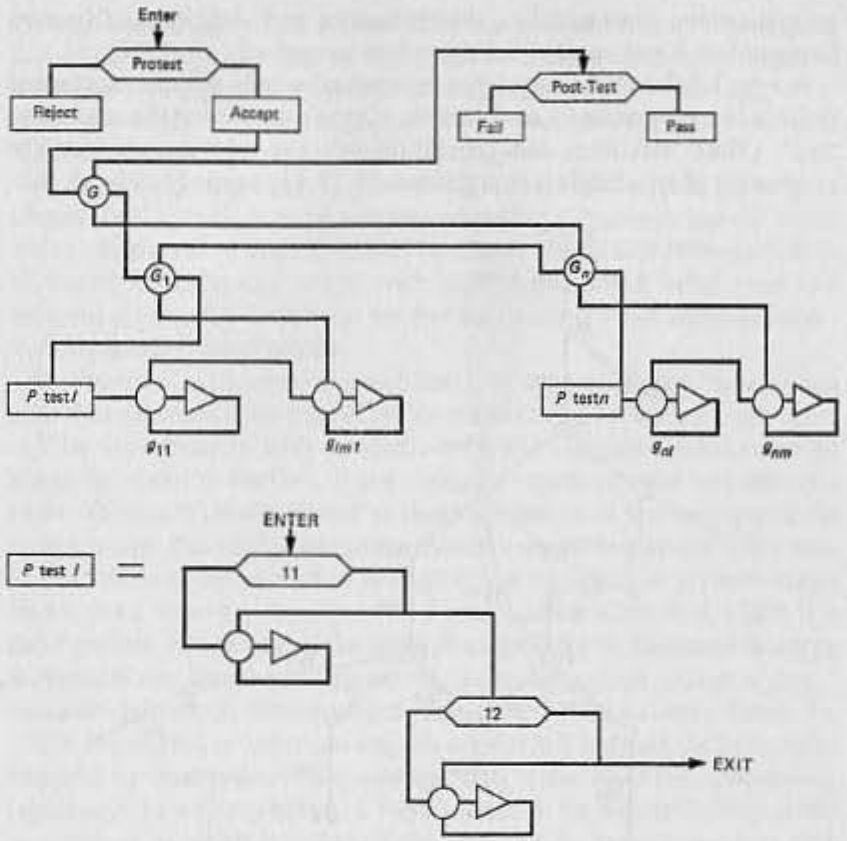


Figure 93 A branching programme format.

5. Consider the first topic of the main topic stated in step (1). It is made of subtopics. Some of these may be elementary concepts and others may be introduced by definitions. Form an ordering of these topics (Fig. 94), starting with the first, in which each subtopic is represented by a node in a directed graph and in which A is above B, if and only if, a concept for topic B must be attained before A is instructed, either a concept for topic B is part of a concept for topic A or the attainment of a concept for topic B is a necessary precondition for attaining a concept for topic A. To each directed edge (directed arc) of the graph, and to each terminal node, assign a type label according to the type of process used in building a concept for the topic at the node where the edge terminates.

So far as the teaching is concerned, a type name is the name of an instruction which the student is asked to carry out, on the assumption he has a *Lev 1* mental operation for doing so. The list of type names is restricted to

generalisation, concatenation, discrimination and definition. *Discovery* (unqualified) is any or all of the operations so named.

A type label may be any 'type expression' where a 'type expression' consists in a type name or combination of type names using the connective 'and'. (Thus 'definition and generalisation' is a type expression.) The assignment of type labels is straightforward. If A is uniquely above B, then

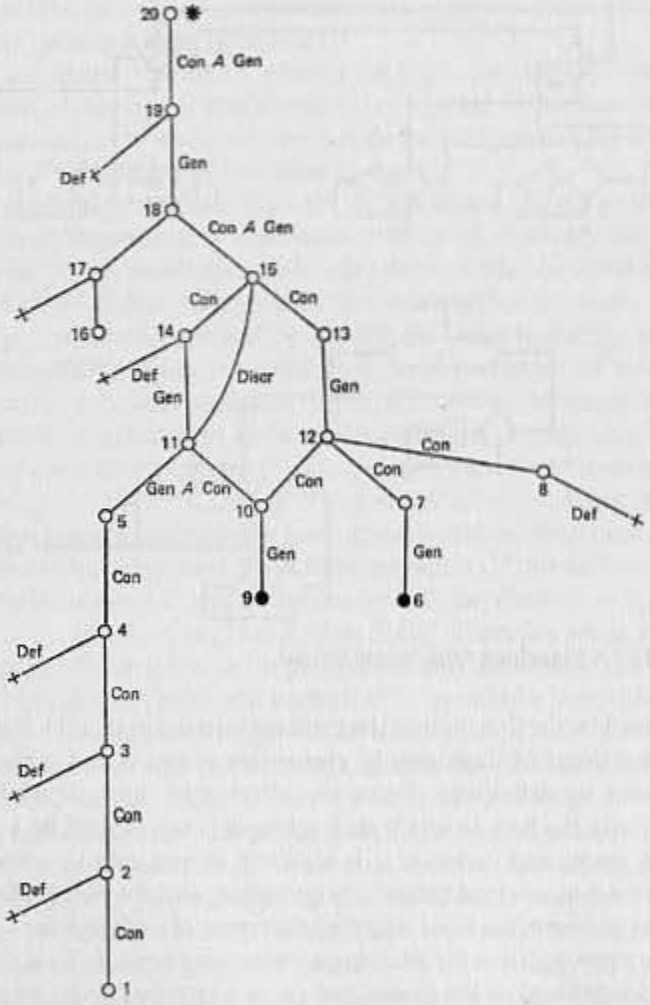


Figure 94 A topic ordering for part of an elementary logic a tutorial programme. The numbers are assigned by a heuristic to indicate sequence of instruction. Type names: Def, definition; Con, concatenation; Gen, generalisation; Discr, discrimination; ○, subconcept node; ●, terminal, node; ⋆, concept at head of ordering; ×, terminal branch for introducing definition.

the edge connecting A to B is labelled by the process whereby a concept of A is obtained from a concept of B. If A is above B and C, then the edge AB is labelled by the process whereby a concept of B is used in building (part or all of) a concept of A, and the edge AC by the process whereby a concept of C is used in building (part or all of) a concept of A.

The topic ordering is closely related to an ordering of skills in the sense of Gagne (1962, 1969). It is also closely related to a 'concept matrix' in the Ruleg programming system devised by Glaser (1962) and Homme (1966). However, a rectangular matrix with topic names which label rows and columns is unwieldy for a large programme, since it is not often necessary to compare all pairs of names.

If subtopic A is above subtopics B and C in a topic ordering, then we can infer that a student having concepts for B and C could achieve a concept for A if he were given suitable instructions (this is Gagne's condition for an hierarchy of skills). Further, at first sight, the organisation of subtopics in a topic ordering is closely related to the organisation of the concepts of the main concept. But on closer scrutiny, there are several important differences.

The 'instructions' referred to in step (2) (the organisation of the concepts into a main concept) form part of a performance algorithm which is a *Lev 0* process. In contrast, if the name of a type label or the names in a type expression are issued as instructions, each instruction brings a *Lev 1* operation into effect; it is an instruction to learn, not just an instruction to do.

The instruction to learn can only be obeyed if it respects the limitations imposed by intermediate data storage. This is the basis for determining legitimacy. In step (6) below, a topic is said to be legitimate only if the instructions to which it will give rise are likely to prove consonant with storage capacity limitations.

6. Suppose it is necessary to learn one concept at once. If so, several different subconcepts must be kept in mind (and by hypothesis, kept available in intermediate memory) whilst others are instanced. It is possible to choose a path (a sequence in which the subtopics are to be instructed) that will minimise the loading by minimising the number of stacked prerequisite concepts that will be created by the tutorial programme based on any one path. In case there is no unique path, a set of 'best' paths is determined. Several heuristics are available for numbering the subtopics and thus delineating the path. They differ slightly according to the assumptions made about interaction in intermediate storage. One of the heuristics is applied and the 'optimally' numbered topic ordering is tested to see whether a numerically valued property, μ, exceeds a critical limit interpreted as the estimated capacity of intermediate storage. If estimated capacity $> \mu$ then the numbered topic ordering is deemed legitimate. If not, then either (i) the choice of topics in the main topic (the choice of BR in BR^*) made at

step 2 must be revised or (ii) it is necessary to introduce repetitions in the instruction of a single topic (which is usually self-defeating; most of them increase the value of μ).

7. Suppose that the numbered topic ordering is legitimate. Using the programme format of Fig. 93, the author (i) writes a test for concepts of elementary topics and (ii) writes a topic statement (goal statement) for topic 1. This does not describe the topic (still less teach a concept for the topic). It must, however, allow the student to recognise whether or not he has mastered the topic after suitable instruction, i.e. whether or not he has achieved the goal of learning a concept for this topic. Further, it shows the student what he and the programme are aiming for. (iii) Write a precondition test, *P*-test 1, covering all the topics other than definitions that are represented by terminal nodes in the numbered topics ordering. (The precondition test contained remedial loops, as indicated in Fig. 93. If the student does not have one of the subconcepts, he is instructed (remedially) until he does have it.) (iv) Taking subtopic 1 in topic 1 the author, using the type labels, writes eight or more sequentially cued, skip linear-frames (possibly interpolated with linear frames) on examples dealing with this subtopic; having stated the subtopic in the first frame. This is essentially the recommendation given in Glaser and Homme's Ruleg programming system augmented by the more recent comments of Davis and Hartley (1972). The process is repeated for subtopic 2 and so on, until the last. (v) A test is written for topic 1 as a whole (by definition of a topic ordering this is a test for concept number m_n). If the student fails in this test, the present format returns him to the start of the programme segment for concept 1. It would be advantageous to return him to remedial material.

8. Groups of students from the target population are matched with respect to verbal abilities and speed of programme reading. Each group of students should exhibit variations characteristic of the target population and the groups are labelled for future reference as *A*, *B*, . . . and so on.

The programme segment for topic 1 is administered to all members of group *A* and their performance is recorded, individually on a record sheet, together with their comments about the material. These comments are of the type 'bored' and 'cannot see where the argument is going' and are of a standard form. Data for the entire group of students is conveniently summarised by a tabulation (Fig. 95) which relates the programme format directly to the topic ordering.

Various criteria can be applied (for example, to secure a mean correct-response frequency or a mean correct-response latency). In Mexico, we regarded the data as a *clinical history* and most of the *diagnosis* of why a *group* did not learn was done by a physician, supported with analytic facilities for searching back along the records and for assessing the records

	Concept name $C1_1$																	
	Subconcept name $C_1 1$									*Subconcept name* $C_1 2$								
Frame number	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88
Frame type	*B*	*A*	*A*	*A*	*A*	*A*	*A*	*A*	*A*	*D*	*C*	*A*	*A*	*B*	*A*	*A*	*A*	*A*	*A*
Mean latency	2	4	4	7	4	6	3	2	0	1	2	3	5	4	5	2			0
Mean correct responses	8	5	7	6	4	2	2	1	0	8	7	5	6	1	2	4	2	0	0
Mean mistakes	0	3	1	2	4	6	6	0	0	0	1	3	2	7	6	4	6	0	0
Mean omissions	0	0	0	2	4	4	7	7	8	0	0	0	0	0	5	7	7	8	8
Comment 1 frequency	0	1	2	0	1	0	0	1	0	0	8	0	0	0	1	0	0	0	0
Comment 2 frequency	0	2	0	0	0	0	0	1	0	0	2	0	0	0	1	0	0	0	0
Comment 3 frequency	0	5	7	0	0	1	0	0	0	0	3	2	0	0	1	0	1	0	0
Comment 4 frequency	0	7	8	0	1	0	0	0	0	0	5	4	0	5	0	1	0	0	0
Concept latency					4									3					
Concept mistakes					22									29					
Mean frames concept					40									45					
Rate of increase of mistakes			0·43			0·57						0·55			0·44				
Comment 1 frequency					5									9					
Comment 2 frequency					3									3					
Comment 3 frequency					3									7					
Comment 4 frequency					16									15					

Figure 95 Recording sheet for tutorial programme data from student group.

of individual students in an attempt to pinpoint specific pathologies. The output from the clinician (or, less effectively, from an analysing heuristic) is returned to the programme author who is required to modify any frames that cause difficulty.

The trick in the scheme is as follows. As a crucial part of step 2, the author approved the topic ordering (either he constructed it or selected and modified an independently constructed ordering). Whether he writes according to this structure is a different matter. But, unless he does so, the data (whatever the judgment criteria employed) will not support his work. Hence, the author is given liberty, at this stage, to change the topic ordering

(though he is naturally loath to undertake such an onerous task unless it is essential). As a result *authors* seem to learn to take the topic orderings they have approved quite seriously.

9. The programme segment is rewritten (in rare cases the topic ordering may also be changed). The rewritten programme is now administered to group *B* of the evaluationary study students and step 8 is repeated. The adaptive process goes on (in practice, not beyond group *C* or *D*) until the basic criteria are satisfied.

10. When the programme segment for topic 1 is complete, steps 6, 7, 8, 9, are repeated, for topic 2 and so on, until all the topics in the main concept have been exhausted.

11. A post test is devised covering the concepts of the main topic as a whole. The only significant difference (as against topic 1) occurs in step 7. Here the precondition test for a topic i, $i > 1$, includes tests for all those concepts established in previous segments that are concepts for topics labelled by terminal nodes of the ith concept ordering.

This adaptive technique is a tutorial operation at *Lev 2*. It is a heuristic. Certain features of the heuristic are readily mechanised (for example, the numbering of the concept ordering) and others can be (for example, the housekeeping details of programme writing could be handled by co-operative interaction with a computer). But some steps, notably writing the frames and possibly finding a good breakdown into subtopics and certainly the clinical analysis, are in the human domain.

3.2 Criticisms of the Programming Scheme The tutorial programming scheme operates quite successfully and, judged by gross criteria, yields effective tutorial programme segments. It does not follow that combinations of these segments will be equally effective because no account is taken of 'long distance' interactions. Consequently when these are important (as they are in learning many subject matters) the result of segment combination is unpredictable.

The main criticisms of the scheme that emerged from using it are more fundamental. As a matter of fact, they apply to all of the schemes based on comparable premises but happen to be well exhibited in the context of this one.

(a) The tutorial programme is a teaching strategy of a specially restricted type. There is only one such teaching strategy and that one may be (empirically often is) out of kilter with the student's innate learning strategies. That is, a student is not, even at the strategic level, a *tabula rasa* open to the inscription of *any* plan for learning, however efficient it may be. Left without guidance he learns using his own strategies, which compete with *some* imposed by the teaching system though they interact cooperatively

with others. The competition can hinder the learning process, and, as judged from our records, may even altogether inhibit it.

(b) The topic ordering is a slightly liberalised form of noun tree; a 'taxonomic ordering' in a very narrow sense. It might be conveniently tidy if knowledge did have such a structure but (fortunately to my mind) there is no real reason to believe it does so, and every reason to believe it does not. It seems that knowledge is structured as a *verb network* or relational network not as a *noun tree*.[3] If so, only *descriptions* of knowledge are hierarchical. For example, though some of our students found it easy to follow the partial orderings of topics, or even the particular sequence delineated by the tutorial programme, others found the task quite impossible. They could only conceive 'truth tables', for example, in many ways simultaneously. To succeed in the learning task they would have needed a large number of (quasi-hierarchical) descriptions of the underlying cyclic structure.

(c) As noted already there is no easy way of evaluating explanations. The multiple choice and constructed response questions that are asked constitute, in the sense of Harrah (1966), *whether* or *which* questions. These can easily be machine evaluated, which is why they are asked in the first place. Our records strongly indicated that the selective replies to these questions (lists of possible selections of one answer from a set of alternatives) furnish inadequate indices of comprehension and that the position is not greatly improved by averaging over responses (it might be by considering patterns of response to an appropriately patterned collection of items).

On the other hand there are firm grounds, theoretical as well as empirical, for believing that a novel or invented correct explanation *does* furnish good evidence for the existence of a concept and that a derivation from, or explanation of, the original correct explanation is evidence for the existence of a concept in long-term storage (my current jargon is different; namely, that the explanation can be reconstructed as a *memory*, preferably in many ways).

3. Take a dictionary and follow through the definition of a *noun*: the successive definitions are nested in a *tree* (the most general the highest and the most specific the lowest). On the contrary the definitions of a *verb* form a network; any verb is defined in terms of other verbs and at some point the original verb is used in the definition of a derived verb. Thus; Push is to Shove *or* Move or . . .; Shove is to Thrust *or* Lean against *or* push . . . (and so on for all branches). The structure is a *verb network*.