be attached to the terms cited and it is also possible, under these circumstances to obtain an index of *correct belief* (Chapter 1). The reality or the relevance of the self-organisation is guaranteed by any scheme that renders any decrease in $H$, which is used as a *control signal* by the regulator, contingent upon either an increase in correct belief or a constancy of correct belief.

In practice, schemes of this kind work very effectively. They have a much wider field of tutorial application than the domain of structured skills discussed up to this point.

# 9 Descriptions of Processes

A description, of any sort, entails a language in which the description is expressed and which can be interpreted by someone or something. For instance, one can describe, in natural language, the events going on in one's room or one's mind, in the street, in an aquarium or in a motor car's engine. The widespread idea that a static skeleton of things must first be described (e.g. the pistons of the engine, its cylinders, crank shaft, and so on), and that events become obtrusive later as happenings to *things* is a misconception.[1] Often this approach is convenient and logicians have achieved formal elegance by adopting it. But there is nothing in man or nature to enforce this world view and (as particle physicists, amongst others, have found) it is sometimes better to notice events first and to build things, either real or fictional, around them.

The outstanding distinction between a static description and the description of a process lies in the capabilities assumed to exist in a user (recall, from Chapter 1, that there is only *potential* information in a description, i.e. literally it *has* a pattern, it does not *have* an information). Both static patterns and processes may represent classes ordered by the writing and connection rules of a language or functions and relations. For example, a table specifies a relation and abstract automata may be represented by a table (for instance, *Ru* of section 1.2 in Chapter 2). But if a pattern is depicted then the processor that uses it must be equipped with an order of execution or a specially operated clock (a seeking-out program) that reconstitutes the table in its own storage and refers to it. If a process is represented (the specification *Rul* in contrast to the specification *Ru*, for example), then order of operation is carried as part of the pattern; it prescribes the process to be executed directly and it may prescribe or permit many execution orders.

---

[1] Little or nothing is said, in this volume, about the description of static objects and patterns; still less concerning the important topic of description, *efficiency*. Without an *efficient* description, most problem solving operations and control operations are utterly impracticable. The reader is referred to Banerji (1970, 1971) for a comprehensive as well as elegant discussion of the field and the most lucid available account of what a description *is*. Perhaps the best technical accounts of pattern recognition, learning, and search are found in Ivahnenko and Lappa (1967) and Fu (1968).

Phrased differently, any function represented in a description may, or may not, have an imperative tag and any relation may, or may not, have a subjunctive/imperative tag. For a process description, functions are to be executed as verbs in the imperative or indicative mood; relations are to be executed as verbs in the subjunctive mood; a pattern description is not to be executed at all but referred to by a process undergoing execution. A process which may, of course, assemble it into a program, just as a reproductive automaton does in accepting a tape description and producing a machine.

## 1 Process Description

A process description can be inscribed (as a picture on paper, as printed words, or in the storage position of a computing machine). It is a *latent* or *potential* process and not the process itself.

Further, if you or the machine have a time sense with respect to past and future there is a sense (*this* sense) in which the process description is an account of *what has happened.* Of course, being equipped with a time sense you or the machine may describe *that* also and by qualifying a process description of *what was* into *what will be* transmute it into a *plan.*

The essential intermediary, required in order to use a plan, is a command and question language; minimally a language that is interpreted and capable of encompassing not only statements but imperatives and interrogations.

**1.1** Suppose, you *intend* to use a plan. If you use it yourself, that plan is a description of your intention, but you might also intend it as a formula for action by some other agent acting in the capacity of an executive. If so, you address the transmuted process-description to the agent as a *prescription.* The agent/addressee may be a person, a group, a machine or an organisation; provided only that the agent is able to interpret the language in which the prescription is written and either to *have* the capability to execute the plan or to be in a position to *obtain* this capability. (In the special case mentioned, you first of all address the plan to yourself.) Equally well, you might use the inscribed process-description, without transmutation, as a *permission giving* structure addressed to an agent or recipient, and allow him to do certain actions or to make certain 'grammatical' utterances in the language (the process description is used as a grammar but this does not mean a syntactic grammar only; it is a semantic and pragmatic 'grammar').

**1.2** The prescription can be represented in various ways. The preferred canonical representation is a command or a set of commands. This usage is uncommonly liberal and requires some qualification. The connotation of 'command' is a statement in the pertinent language of the type.

(a) Addressee ! Do $P$/Given $X$

or (b) Addressee ! Bring about $Y$/Given $X$

Where $P$ is a process, and $X$ is a precondition (the command to do $P$ being obeyed by the addressee if and only if $X$ holds); $Y$ is a further condition often called the goal condition. In fact (a) and (b) are equivalent in so far as doing $P$ brings about some $Y$ and achieving $Y$ involves the execution of some $P$.

Henceforward, we shall treat (a) and (b) as equivalent with a bias towards (b). The form (a) is a specific *instruction*, a very special type of command, like 'add' or 'move on' and it says nothing about the result of obeying the command. It is characteristic of run-of-the-mill, particular-to-general, programming languages. In contrast, type (b) invokes the achievement of $Y$ (given $X$) by any means at the addressees disposal, and it presupposes the existence or constructability of $P$ for this purpose. Type (b) is characteristic of general-to-particular programming languages like PLANNER or MICRO-PLANNER (Hewitt, 1969, 1971; Winograd, 1972).

**1.3** By comprehending the complement of the precondition, the command form becomes the conditional statement with fixed addressee

'If $X$ then $Y$ if not $X$ then nothing'

which is readily extended using a command disjunction of Rescher (1966) (one, and one only, of $Y_1$ or $Y_2$ depending upon data to be supplied) to yield the standard conditional form

(c) 'If $X$ then $Y_1$: if not $X$ then $Y_2$' $\triangleq$ 'If $X$ then $Y_1$, else $Y_2$'

or in general to yield the complex conditionals

(d) 'If $X_1$ then $Y_1$ else if $X_2$ then $Y_2$ else . . . if $X_n$ then $Y_n$*'

However, the command form is open to fuzzy interpretation, which is essential in the developments of the next volume, as are any of Zadeh's (1973) fuzzy conditionals.

**1.4** New commands may be composed and decomposed (ultimately to instructions) using either Rescher's (1966) calculus or a mild extension of it in the coverage of which command graphs are introduced as well as Rescher's command chains. In this sense any prescription for a process is representable as a command, or a set of commands, but certain variants are very important.

1. A command (large or small) may, as a special case, be *unconditional* (⟨Addressee ! Do $P$/for any precondition⟩ or ⟨Addressee ! Do $P$/One *state*⟩ and *that* state holds).

2. If the relation between $X$ and $Y$ is a *function* then the command is interpretable only as a direct *imperative*. It gives a value in the range of the function for a value in its domain.

3. If the relation between $X$ and $Y$ is not a function, then the command is only interpretable as a subjunctive imperative.

4. A command that is obeyed is a *goal*, in the sense of an *intention*. It may be that a *goal condition* ($Y$) is itself a goal (this is always true, for example, in my theory of conversation and learning).

5. A command to make an utterance of a given form in the language, *as though* a condition held, is a *question*.

6. A question may demand an explanation; if so, the explanation is a description of the execution of the original command (to bring about a goal condition) in a context that is generally left free.

Given (4) above, the execution of a command *models* a relation and the *explanation* is the description of a certain modelling operation (as required for example, by Loefgren, 1972).

7. Either the command or its question variant may be qualified; the qualification restricts the context in which an explanation is given.

8. If the linguistic form of the answer to a question is a statement satisfying an exclusive disjunct, it is a *whether* or multiple-choice question; if several such statements are permitted (given a list of disjuncts) or if there is an open disjunction (asking for a *constructed* response in reply), then the question is a *which* question. The logic of the syntax of *whether* questions and *which* questions is developed by Harrah (1966). His recent treatment of more general questions 'Erotetic Logic' (Harrah, 1973) differs from the present approach but is compatible with it.

**1.5** In overview, a process description may be converted into a plan. Again, it may be addressed to an executive agent; if so the process may be *prescribed* or *permitted*. In either case its execution, or possible execution, may be imperative/subjunctive, imperative or interrogative. The verbal summary is shown as a graphic condensation in the chart.

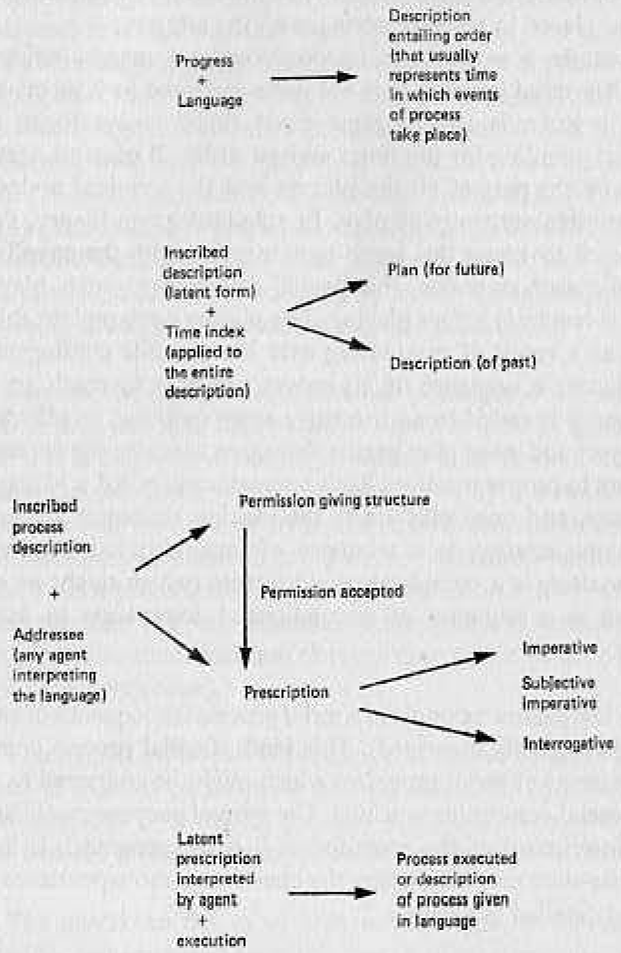
In this picture it is also supposed that the agent *has* the wherewithal to execute the commands, but all we require is that an agent should either *have* the wherewithal or should be able to obtain or construct it. Hence, two further distinctions are needed, namely:

($\alpha$) If an agent has the wherewithal and obeys a command, then it is a command to *do*.

($\beta$) If an agent does not have the wherewithal and obeys a command, the agent may or may not interpret it as a command to *learn* to do. Certain agents, notably human beings, act like this by design.

($\gamma$) A command to do certain operations that bring about the wherewithal for obeying (other, as a rule unrelated, commands) is a command to learn.

As before, the command may be an entire prescription or any part of it.


Progress
+
Language
Description entailing order (that usually represents time in which events of process take place)
Inscribed description (latent form)
+
Time index (applied to the entire description)
Plan (for future)
Description (of past)
Inscribed process description
+
Addressee (any agent interpreting the language)
Permission giving structure
Permission accepted
Prescription
Imperative
Subjective Imperative
Interrogative
Latent prescription interpreted by agent
+
execution
Process executed or description of process given in language

## 2 Execution Order

Any process whatever entails the notion of execution *order* (that is, what distinguishes a process description from a static description). The following kinds of process order, though not exhaustive, are particularly important; namely, serial processes and non-serial processes.

**2.1** A serial prescription is a chain or list of commands, conditional or unconditional. The process obtained on executing a serial prescription, i.e.

a *serial process*, is exhibited using one token which resides in one, and only one, position at once (a 'locus of control' or the command currently undergoing execution). Thus a record of the conditions brought about is a list or sequence of items that can be tagged, like the successive positions of the one token, and placed in correspondence with the integers.

For example, a sequence of unconditional commands (either to *do* or to *learn*) is a serial process. It is not quite so trivial as it seems to be. For example, a game is usually represented (in extensive form) as a tree, with nodes standing for the states arrived at by all possible combinations of moves on the part of all the players and the terminal nodes standing for the possible outcomes of play. In standard game theory, the players are supposed to know this game tree, together with the payoff function; that is (for each outcome, the payoff delivered to each player *if* that outcome is reached) before playing. The players contemplate this description and, as a result of ruminating over the possible contingencies, each player chooses a sequence of *his* moves sufficient to reach an outcome. This sequence is stated to an executive agent (without revealing it to any other player) and once play begins the agent executes the sequences of all the players in proper rotation. Such a sequence is called a *playing strategy* in the game, and once play starts the playing strategies are immutable, i.e. a playing strategy is a sequence of unconditional commands; so, for that matter, is a completely rigid lecture (which might be charitably interpreted as a sequence of unconditional commands to *learn* rather than *do*).

**2.2** A less narrow example of a serial process is a sequence of conditional commands (possibly questions). This kind of serial process comprehends the whole range of serial programs which might be addressed to a (general purpose) serial-computing machine. The general purpose machine, together with an inscription of the prescription (i.e. the program), is isomorphic with a finite-state machine, under the one-to-one correspondence '⇔' given below and detailed in Chapter 2.

| | |
|---|---|
| Initial state, input state sequence | ⇔ Fixed programme data |
| Internal states | ⇔ Values of programme variables |
| Output states | ⇔ Values of output variables |
| Steps in execution | ⇔ Steps of sequencing shift register |

**2.3** If the program data is not fixed and is delivered, in an appropriately synchronised manner, during execution, then the finite-state machine has a free input and the computation need not be fully determined. In general, however, a fixed input is assumed and, if so, the program is equivalent, in the abstract, to a finite automaton.

This equivalence is usefully referred to the game formulation. Whereas standard game theory represents playing strategies as sequences of unconditional commands, the development of game theory by Banerji (1970) encompasses the notion of a strategy that is conditional upon the moves of other players. It is still true that once playing begins, the players (having chosen their strategies) are no longer in control. But they have specified, as strategies, a set of finite-state machines (serial programmes) that are executed by an executive agent. The entire game is represented by a directed graph (not usually a *tree*) and the playing is the action of a large finite-state machine. The individual, strategic, finite-state machines may thus be cognisant of any game states the players have anticipated before the play and may utilise *feedback* data. In fact the Banerji game is derived as a special case of *feedback* control (a control operation *is* a playing strategy) whereas standard game-playing strategies mediate feed-forward control only.

**2.4** Suppose the conditional commands in a sequence are subjunctive in form. If it happens that the executive agent is a *serial* general-purpose machine (i.e. it can only deal with one locus of control at once) then the permissions, possibilities, or enablements given by the subjunctive interpretation can only be resolved by (more or less complex and precondition biassed) random selections. Hence the serial execution of the subjunctive imperatives in a prescription is biassed-random and the general-purpose machine, together with an inscribed prescription is isomorphic with a probabilistic finite-state machine (of a type determined by the complexity of the biassing dependencies).

**2.5** Finally, an *algorithm* (Markov, 1961 is the original and clearest reference) is a serial prescription and its execution by a serial general-purpose machine is a serial process; the general-purpose computer, together with the inscribed prescription is isomorphic to a Turing machine.

**2.6** The moral, so far, is as follows. Prescriptions contain a certain order; in the cases examined up to this point it either *is* or *it can be interpreted as* a serial order. Any agent (e.g. a processor) that executes the prescription also has an ordering principle; for example, that it executes one command at once (the command on which our single token is placed at that instant). Phrased alternatively, the engine has one locus of control at once. Moreover, its *state* is always well specified. There are many reasons why an executive agent may be unable to obey a prescription; for instance, because it cannot interpret the language in which the prescription is written. A consideration that overrides all of these minutiae is that the order in the prescription shall correspond to the ordering principle in the executive.

**2.7** The other category of processes, non-serial processes, is a mixed bag. A non-serial process contains junctures at which the order of execution is irrelevant. If it happens that simultaneity involves no *conflict* (that is, the execution of one command does not preclude the execution of another) then execution of the process can be represented by placing several tokens on positions that stand for commands and moving these tokens *independently* as the commands are obeyed. Holt (1968) refers to non-serial processes of this type as 'occurence systems' which is the best canonical form for representing 'parallel computation' of one kind or another. The prescription for a non-serial process of *this* type can be accepted by any executive agent that comprehends the language; provided that it is made up from sufficient small processors, which act independently but are fully synchronised. Because of that, the combination of processor and inscribed prescription is characterised by as many loci of control as there are marker tokens but it has a well defined state at any instant of execution. Further, the loci of control do not interact though the supervisory synchronising mechanism will combine the separately computed data at instants determined by the prescription. The performance of such systems has a general interest; it represents reality quite well and parallel processing saves time and occasionally space. But nothing very new emerges whilst the control loci remain independent. The most penetrating analysis of parallel systems is given by Minsky and Papert (1969), in the context of a special class of pattern recognising machines (the more elementary versions of Rosenblett's 1961 'Perceptrons', Chapter 2, Fig. 196).

**2.8** This type of non-serial process devoid of conflict, has an execution *tree* rather than an execution *chain*. Several tokens could be moved down the tree to represent simultaneous command executions. For example, several property tests can be evaluated at once and variables may be assigned several values, thereafter isolated and separately treated. Perhaps the most significant difference between the serial and the (restricted) non-serial process is that a subjunctive imperative command may lead to several results (studied as distinct, however, unless there is a directive to coalesce them in a specific way). Whereas, in serial execution, these commands are necessarily resolved as one value by a metalinguistic ranking or maximum finding operation (Shimura, 1973; Zadeh, 1973) or to yield one alternative (by a biassed random selection).

Non-deterministic programmes (in the sense of Manna, 1970) are prescriptions of this type. So are the fuzzy algorithms of Zadeh (1973). If executed by a serial device, the conditionals lead to a numerical solution signifying grade of membership in a fuzzy set; whereas, if executed in parallel the fuzzy members of the fuzzy set, or its extremum, form a non-numericised solution).

More interesting possibilities appear if a suitable non-serial prescription is addressed to an executive agent having several loci of control that are not necessarily independent. These loci are most conveniently regarded as distinct but initially asynchronous processors that can be drawn into synchronisation in so far as the execution of a command by one of them interacts with the command being executed by another. Since the processors are not synchronous, there is not always a *state* of the entire system. Moreover, the cooperative and synchronising interactions between loci of control amount to (non-trivial) information transfer. Subjunctive imperative commands, in particular, give rise to this type of interaction (the simultaneous execution of fuzzy conditionals, where the fuzzy solution from one control locus is the input to a computation at another locus). Let us call the process obtained by executing this type of non-serial prescription a *concurrent* process, as was the case for machine processes of Chapter 2.

## 3 Some Defects

The following concurrent process descriptions all have certain defects (to be discussed in a further volume) as they are tailored to suit very different applications. Combinatonic networks (Barralt-Torrijos and Chiarviglio, 1971; Chiarviglio, Poore and Barralt-Torrijos, 1971) derived from a combinator logic approach to computation, represent both parallel, serial and concurrent processes. Petri nets (Petri, 1965; Holt, 1968) are slightly more general but correspondingly intractable. Dienes (1972) has discussed the matter.

One concealed (some say 'obvious') assumption underlies all of these prescriptions and processes. Whether serial, *parallel*, or *concurrent*, the process has a *beginning* and an *end*; at these points any processor does have a well defined *state*.

Personally, I do not find it at all obvious that all processes *do* have beginnings and ends. 'Process' could be defined as *having* this property, of course, but many applications of the term 'process' to real activities would be prohibited as a result of such a definition. I shall try to justify this point as well as illustrating the other distinctions by reference to executive agents less restricted than most general purpose computers; namely, human beings who compute with complex- and variable-order principles.