EWD 196-0

An effort towards Structuring of Programmed Processes.

0. Introduction.

It is my impression that one of the most deep-rooted difficulties in programming is to bridge in one's imagination the gap between the (static) program text and its (dynamic) behaviour, between the algorithm and the process described by it. This would be considerably easier if processes would present structural properties closely reflected in the structure of the program text.

Methods to structure program texts have been widely explored by syntactical investigations over the last few years. The question remains: what process structures to be represented in the program structure are we looking for. In exploring those I have used two yard sticks: our ability to understand and our ability to implement.

In looking for useful process structures I have started from a multiprogrammed environment. I have done so for two reasons. Firstly we may expect computers of the future to exhibit more and more parallelism and it will be vital that a configuration can be extended with, say, another processor without upsetting the whole software system. Secondly, if we indicate to what extent two disjunct tasks can be delegated to two temporarily disjunct computers, we have also indicated to what extent these processes are logically independent, thus guaranteeing a non-interference from which the clarity of the process can only benefit.

Actions, processes and states.

It is customary to regard a sequential process as a time succession of actions, each of which has a well-defined effect. The total effect of the process is then the cumulative effect of the successive actions. The first subdivision of a process into actions may be a coarse one; then the composing actions themselves may again be regarded as processes, i.e. being thought of as composed of a time succession of subactions, etc.

Disregarding input and output we can describe the effect of each action as a "change in the current state of the process", the current state at the end of the action – and thereby the effect – being a unique function of the state at the beginning of the action.

If we use values of variables to record the current state we can regard the variables as coordinate axes in a "state space", with as many dimensions as variables. The current state is then given by a point in state space; furthermore the metaphore of "a subspace" is offered to us when we consider an action that is only concerned with a subset of the variables.

The above picture is reasonably adequate when we are concerned with a sequential processes operating in a constant state space. If these conditions, however, are not satisfied, the picture is badly in need of refinement.

I mention two reasons. It is all right to regard an ALGOL inner block from the point of view of the embracing block as "an action". As a whole its effect is defined by and reflected in values of variables

specifying the state of the embracing block (or blocks). But as soon as we regard the inner block as a process in which the local variables of the block have to be taken into account, we discover that this change in our attitude is accompanied with a temporary extension of state space, viz. with a new dimension for each local variable.

Secondly, we assumed the possibility to speak about "the state at the beginning of an action" and "the state at the end of an action," and thus assumed an implicit definition of simultaneity, covering all the variables and specifying the moment at which their value should be taken to specify "the current state". This condition is certainly not satisfied as soon as our process has been split up over two or more parallel processes, that may have been delegated to as many mutually unsynchronized computers! In that case the effect of actions is only defined by and describable in a projection a subspace: the other dimensions have to be ignored.

Before trying to refine the traditional picture I should like to make some remarks about it.

Firstly As long as we regard a sequential process as a time sequence of actions, leading from one state to the other, then the current state is only defined in between those actions. During an action the current state is undefined, or better perhaps: the whole concept of "current state" is inapplicable. To regard an action as a process composed of subactions is nothing more than a means

to bring a finite number of intermediate states in their perture. In other words: there is no point in talking about the current state of an action. the notion of "the current state" is applicable only to a process composed of actions, and what is meant by "current" is related to "... (a finite number of) discrete points of progress, marked as "after the preceding action but before the next."

Secondly in real time these discrete points of progress may be periods of time, viz. when a process is temporarily stopped in between two successive actions. If the notion of "current state of a process" is to be of any use at all, we must be sure that in such a period of time between two actions the current state remains constant in time. In other words, if the current state of a process is recorded by the value of variables we must be sure, that no actions of other processes will change these values: the variables used to record the current state of a process must in some sense be "private to the process", each process is related to its own subspace.

Thirdly, I have come to the conclusion that the usual concept of a sequential process needs some generalization: in a sequential process the next action only begins after the preceding one has ended: this, however, implies that no two actions of the same process can ever overlap in time.

"Thirdly I have come to the conclusion that the usual concept of a sequential process embodies two different aspects and that there is a point in retaining only one of them. In a sequential process the next action only begins after the preceding one has ended: this however implies that no two actions of the same process can ever overlap in time. The two different aspects untertwined are: explicit sequencing of actions on the one hand and "mutual exclusion in time" on the other. The latter aspect seems the more fundamental one: only if actions exclude each other mutually in time we can attribute a time sequence to them.

We shall admit a sequential process "to get unsequenced" for a finite period of time, i.e. during that period of time it is still composed of actions that exclude each other in time, but the order in which they occur - or stronger even: if they occur at all - is for the time being no longer an intrinsic property of the process.

We shall give you a clear example of this situation, viz. the solution of the "Problem of the Dining Quintuple."

[Insertion].

"We can distinguish in this example six different subspaces: the five local spaces of the persons concerned - the spaces in which "eating" and "thinking" is described for each of them - and the global space of the state variables and the semaphores.

The five personal processes are only initiated and the corresponding subspaces are only created at the entry of the parallel compound. The global subspace remains in existence and we have indeed actions taking place with regard to it: the critical sections in the person behaviour, mutually excluded by the P and V-operations on the binary semaphore called "mutex". We regard them as actions of the global process; they only take place (however) sequenced in the personal processes.

Allowable modifications of state space.

The actions discussed so far bring about a change of current state: the current state before and after are described with respect to the same state (sub) space. The dynamic introduction of new variables - such as at block entry - yet falls outside the picture. Let us try, and let us start with the ordinary inner block of ALGOL 60, with only local scalars.

Many implementers have remarked that if we allocate scalars in a stack, that then, according to the distance from the stack bottom, the local variables of successive inner nested inner blocks can be numbered consecutively, corresponding to the distance from the stack bottom.

Many implementers have remarked that if we allocate scalars in a stack, then the local variables of nested inner blocks can be numbered consecutively by the translator, while at run time this number can be related to the distance from the stack bottom.

That is: the variables belonging to a block and its inner blocks can be identified as indexed elements of a vector, this vector ~~decreasing~~ increasing and decreasing in stack fashion. And also: if this block operates on these variables only, only the identity of this vector has to be implicitly understood. Or again: the total execution of an inner block is an action of the process described by the embracing block. The inner block can be regarded as a process, but then operating in an other (be it related) subspace. In the middle of the execution of the inner block, the state of the outer process is an inapplicable notion!