Determinism and recursion versus non-determinism and the transitive closure.

Two years ago I pondered about the design of an intellectually well-manageable programming language, the implementation of which would allow a potentially very high degree of concurrency without imposing it (nor, of course, requiring the amount of temporary storage that would be needed to simulate the concurrency.) One can try to reach such a goal by the introduction of multi-component datatypes like in APL, but then the price to be paid seems to be a very elaborate repertoire of operations, and, having seen where that leads to, I obviously did not want to go down "that slippery road of reasoning".

I found myself considering the massaging --under control of a very limited repertoire-- a set of n-tuples, the possibility of concurrence being modelled by the simultaneous creation of a whole lot of such n-tuples. Wanting to be kind to my hardware friends, I focussed my attention at a very early stage to the question whether I could fix n . Extremely far-fetched analogies suggested that if n could be fixed, it should be fixed at 4 , and as a result the n-tuples became 4-tuples and were tentatively called "quadrons". In a next stage my (originally hardware) friend C.S.Scholten came with an argument that, whatever could be done using n-tuples could be done --after a fashion equally well-- with k-tuples, with $k = \text{entier}((n+1)/2)+2$ After this encouraging confirmation of my hunches, we got stuck completely and the exercise was dropped for at least one-and-a-half year.

A number of months ago, the exercise was picked up again by W.H.J.Feijen and M.Rem (with some stress on the latter) and occasionally me. Analyzing the previous failure we came to the conclusion that the fixation $n = 4$ had been premature, so we dropped it and from then onwards called our n-tuples "associons". We tried again --and some of our efforts that need not concern us here, have in the mean time again been unmasked as dead alleys--, and one thing seems to have emerged more or less solidly: about our most central primitive operation can be viewed as forming the transitive closure, i.e. given a set of nodes of a finite, directed graph as "starting set", determine the set of all nodes reachable via a directed path from at least one of the nodes of the starting set.

* * *

The incentive to start the experiment two years ago was only partly a desire to explore the programming of potentially high concurrency, it was also the result of my difficulties in writing a --"sequential", i.e. ALGOL-like-- program that should determine the convex hull of a (large) set of given points in three dimensions. For the convex hull in two dimensions I knew a host of algorithms; I tackled the three-dimensional version of that problem in the expectation that from that exercise I should be able to learn something about programming methodology. The first thing I learned was that I could not solve the problem nicely: what I tried became so messy --at least to my standards-- that I aborted each effort long before I had reached a working solution. So I dropped that problem for the time being --apparently not ripe for it yet--and started to write a book instead.

The problem of the convex hull in three dimensions presents some difficulties which are absent in the two-dimensional version: it is not patently obvious how the result --and, therefore, intermediate states-- should be represented in an (essentially) "linear" store, nor is it patently clear, how the two-dimensional surface should be processed by a "sequential" machine. Wanting to include in my book a true and non-embellished design history, --including the risk of failure--, I needed a non-trivial problem I had never successfully solved before. I choose the problem of the convex hull in three dimensions, and, after having reported what I had learned from my non-successful attempts, I started to try to solve the problem in earnest. I succeeded --as a matter of fact: I arrived at a solution which filled me with some pride-- and my solution had two characteristics which seem worth mentioning. The one observation is that the need to introduce more-dimensional arrays emerged nowhere; the second observation is the central role played --at two/three places, in different disguises-- by the task to determine a transitive closure. More precisely:

Given a finite set S and a subset B; for each element x of S, zero or more other elements of S are by definition "the consequences of x". Determine the set V, defined by

1) each element of B belongs to V;
2) if x belongs to V, so do its consequences
3) V contains only elements that belong to it on account of 1 and 2.

The pattern of my program was as follows:

```
C := B; V:= empty;
do C ≠ empty → chose an arbitrary element  c  from  C ;
               transfer  c  from  C  to  V ;
               extend  C  with all consequences of  c  that do not
                  belong to the (disjoint) union  C + V
od  .
```

It is worth noticing that the intersection of C and V remains empty and that under the assumption that the time needed for the test for set-membership does not depend on the size of the set concerned, the time taken by this algorithm is absolutely independent of the choice for c : at each repetition the set V is extended with one new element.

In the case that the graph is a rooted tree, whose root node is taken as B , the problem of finding the transitive closure is also known under another name: traversing a tree. Its standard solution is a recursive one, stronger: the recursive solution for tree-traversal is often regarded as the prototype application for recursive procedures. If we compare the recursive tree-traversal with the above, more general, algorithm for the transitive closure, we see that the role of C is taken over by the anonymous stack. This can be done, because in the case of tree-traversal the test, whether consequences of c are members of C+V can be suppressed: the absence of cycles in a tree guarantees that we shall never "meet" the same node more than once.

In my innocence, I had always regarded this recursive solution for tree-traversal etc. as a, in some sense, basic and fundamental algorithm, and for years I have taken the central role of anonymous stacks for granted. It was a little bit of a shock for me to discover that the anonymity of the stack is only allowed, provided that we restrict ourselves to a rather specia case of a more general (and perhaps in its general form more "fundamental") problem, viz. the transitive closure. It made me wonder, whether in the last fifteen years --since LISP and ALGOL 60, say-- we have perhaps over-emphasize recursive solutions, and have taken the last-in-first-out strategy for choosi c --often purely a matter of convenience, as far as storage management is concerned!-- so much for granted, that the rediscovery of the freedom to select an arbitrary element c from C could rank as a scientific discover of some sort.

The above may shed some light on current linguistic exercises in artificial intelligence work. A last-in-first-out queueing discipline engenders --as every operating system designer knows-- the danger of individual starvation, and if the purpose of the algorithm is to travers a finite portion of a potentially infinite tree, the last-in-first-out stragey is clearly unacceptable (whether other strategies will lead to acceptable performance is a question that falls outside the scope of this note). When, however, these linguistic exercises give birth to some system of recursive co-routines, should we not then raise the question, whether such exercises are not drastic enough, are only half-hearted attempts at freeing ourselves from the shackles of recursive solutions, which, perhaps, have dominated the scene already too long? (If the mere suggestion that recursive solutions may fail to be a cure for all our problems, don't blame me!)

The question, among other things, hinges upon the assumption that the problem of the transitive closure for directed graphs in which arrows may merge, is --besides being a logical generalization of the tree-traversal-- a sufficiently frequently recurring theme to give it the status of "central problem". I cannot collect all evidence in the world, but I can give you some. Last Saturday morning the postman delivered two reports at my doorstep: "A case for the <u>for</u>-statement" by Edouard Marmier (ETH Zurich) and "Constructing correct and efficient programs" by M.Sintzoff and A.van Lamsweerde (MBLE Brussel and I had a pleasant weekend studying them. When Marmier has remarked that for the sake of robustness, implementations should enforce a certain requirement --one of non-interference, essentially-- he ends that paragraph by observi that even without additional information in the program text "...the problem of enforcing the requirement is only of moderate difficulty, since it reduces to the problem of forming the transitive closure of a directed graph. For this, a nicely implementable algorithm exists." The major part of the report by Sintzoff and van Lamsweerde is concerned with the problem of reducing the number of dynamic re-evaluations of synchronizing conditions. But the whole problem of determining, which sleeping processes can be woken up is one of determining a transitive closure! (A process goes to sleep when execution of a critical activity would cause violation of the imposed invariant relation. Such a critical activity remains pending until, after completion of another critical activity, it is detected that it could and decided that it should be fired. Such a secondary critical activity may enable other pending ones

concurrent

to proceed and the waking-up should continue until the fired critical activities have caused a state in which none of the still pending critical activities is kept pending without justification: only then the waking-up obligations have been fulfilled. The waking up has exactly the same cascading nature --with "merging arrows" included-- as the detection of the transitive closure of a directed graph.) Was it entirely accident, that these two reports were dropped on my doormat simultaneously?

* * *

While pondering about all this, it struck me that there is another area in which we encounter rooted trees, viz. the states of terminating computations in a (finite state) <u>deterministic</u> automaton: if we so desire, we can distinguish a forest of trees, with a one-to-one correspondence between terminal states and (roots of) trees. The question of the weakest pre-condition for terminating in a given final state amounts to traversing the <u>tree</u> with that final state as its root node. But consider now the non-deterministic automaton, in which some states may have a set of possible successor states. From a very operational point of view, one may feel forced to introduce some "ghost-input", because, without it, the automaton would not "know" which way to go: logically, however, this amounts to squeezing the non-deterministic automaton into the straitjacket of the deterministic one.

(In the process of mathematical discovery, the introduction of "something", from which then later can be "abstracted" is an utterly respectable one. But, once such an abstraction has been discovered, from a methodological point of view, I always feel that then the concept should be given its independent right of existence if it is to bear fruit. First Descartes introduces coordinates by choosing the axes arbitrarily; it is when the arbtrariness of this choice is fully realized, that coordinate-free methods are born, and I hope never to forget my excitement when I saw the following proof (from the lectures of J.Haantjes) of the fact that the perpendiculars of a triangle go through one and the same point. It defines H as the intesection of two perpendiculars and shows that the line through H and the third angle is also a perpendicular;

$$\begin{array}{ll} \text{given:} & (a - h) * (b - c) = 0 \\ \text{given:} & \underline{(b - h) * (c - a) = 0 \quad +} \\ & - (c - h) * (a - b) = 0 \qquad \text{Q.E.D.} \end{array}$$

The usual high-school proof is by tortuously constructing another triangle, of which the original perpendiculars are the bisectors. So much for the fruits!)

The removal of the restriction to deterministic automata means that our "backward scan" can no longer be viewed as traversing a rooted tree, because such backward paths are now allowed to merge. (In this discussion it seems irrelevant that, if we want the weakest precondition it is not exactly the predecessor relation that we want the transitive closure of; we have something like
1) all states satisfying R are in V
2) each state whose successor set is in V , is itself in V as well.
The important thing is that we have departed from rooted trees.)

The methodological question that I am tempted to raise is the following: could the problem presented by generalizing the current theory of denotational semantics so as to cover non-determinism as well, be related to a preponderant role of recursive definitions, in a way "pushing" trees where more general graphs are needed?

* * *

A next question to which I may need the answer by the time that I can formulate it precisely, is the following. Assuming that the formation of some sorts of transitive closures do indeed play the fundamental role that I currently do not exclude --all evidence in either direction would be most welcome!-- and assuming that a "machine code" can be designed in which such operations are regarded as primitive, while we hope that they can be implemented by highly associative techniques, involving concurrent activity "all over the place"; assuming further, that such a machine code suggests solvability of a problem in an orders of magnitude less exploding number of steps than today's complexity theory tells us. It certainly implies that my "machine" will be hard to build; will it also imply the impossibility to do so? I just don't know, to what extent the results of complexity theory can be carried over to unusual techniques. For instance, finding the shortest connection between two nodes of an undirected graph with positive edge-lengths is polynomial or something in the number of nodes and/or edges. How much of that argument, however, is applicable to the analogue device, that is made by replacing the edges by gas tubes of proportional lengths and applying a voltage difference between the two nodes? The spark will choose the shortest path. In polynomial time...?

If the above is clarifying or inspiring for any of its readers, I am glad that he saw it. If it evokes possibly helpful comments, I shall be glad to receive them.

14
~~10~~th October 1974
Burroughs
Plataanstraat 5
NUENEN - 4565
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow