On avoiding the infinite.

This chapter should be skipped by most of my readers: it should be skipped by all those that could not care less, it should also be skipped by all dyed-in-the-wool logicians, who probably care too much. Its inclusion is the consequence of a few piercing remarks made by John C. Reynolds in earlier correspondence between him and me. It goes without saying that he is in no way to be held responsible for the following exposition, which, I am sure, is somewhat shaky anyhow.

With respect to modelling what computers can do for us, there are two extremist views.

The one view stresses the fact that computer stores have only a finite capacity and that therefore our machines can be regarded as a finite state automaton. This has as a consequence that if the computation proceeds without terminating , it must return within a finite number of steps in a state in which it has been before. If the machine is fully deterministic, history will from then onwards repeat itself, and the computation will therefore never terminate. If the machine is non-deterministic in the sense as we have introduced non-determinacy --i.e. that in some states there is a choice between a finite number of alternatives-- the situation is more complicated. We may have states where the machine will never terminate, we may have states where the machine will certainly terminate and we may have states where the machine may terminate after an a priori unbounded number of steps. In a machine with two boolean variables "stop" and "go on" we can investigate the following program:

```
do go on → skip
 [] non stop → skip
 [] non stop → stop:= true
od
```

If "go on" is true, it will remain so and proper termination is excluded. If "go on" is false, we have to distinguish between two cases: if "stop" is true, termination is ensured, if, however, "stop" is false, the non-determinacy leaves the choice between the second and the third guarded command open. As long as the third guarded command is not selected, "stop" remains false and the second command may be selected an unbounded number of times: termination is not guaranteed but it remains possible.

The other view stresses that, although the number of states of a modern computer is indeed finite, it is in practice so incredibly large that the remark that the deterministic machine is bound to start cycling is rather irrelevant, because the length of most cycles is such that we could never live long enough to see the machine perform such a cycle. The freedom to forget about the fact that we can only distinguish between a finite number of different states is then immediately exploited by introducing variables of type integer whose range --all whole numbers-- is clearly infinite. In our previous examples we have very clearly done so. How does this model, however, relate to what can take place in our admittedly finite computers?

The extremists with the second point of view forget that nobody but the Good Lord could make such a machine --and that up till now He has failed to do so!-- and have made it a subject of intensive study, to the extent that they have pondered about what it could do when we would set it in motion from now until eternity. As soon as one starts asking oneself such questions, however, its infinite size is no longer only a luxury but also creates problems, problems we should like to avoid. In this monograph we try to steer a middle course between the Scylla of the finite constraints and the Charybdis of the unfathomed infinite. We shall try to come away with just enough theory for dealing with the behaviour of the unlimited machines only for initial states such that proper termination is guaranteed.

Let us first restrict ourselves to the simpler case that the unlimited machine is fully deterministic. In each initial state activation of our machine will then give rise to a unique happening; if that happening terminates after a finite number of computational steps have taken place, a finite number of integer values have been manipulated and therefore their maximum absolute value had a unique lowest upper bound. We repeat that that lowest upper bound is uniquely determined by the initial state. For that initial state we apparently did not need our unlimited machine! The computation could have been done by a machine of finite size, and even stronger: we --or at least the Good Lord-- could decide upon a sufficient size a priori.

The critical step in the above reasoning is that the knowledge of an upper bound for the number of computational steps allows us to conclude an upper bound for the maximum absolute value of integers to be catered for. Let us now consider a non-deterministic machine and the class of

of possible happenings that may take place when we activate it in an initial state such that only properly terminating computations can ensue. Each individual computation of that class will only manipulate a finite number of values, but if the class of possible computations is infinite, a maximum abolute value manipulated is no longer necessarily bounded. That would be nasty and we should try to convince ourselves that the form of non-determinacy we have introduced is so mild, that infinity of the class of possible computations when termination is guaranteed, is excluded. Luckily we can, if the *if* ... *fi*- and the *do* ... *od*-constructs are our only source for non-determinacy. Firstly each guarded command list contains a fixed and finite number of alternatives. Secondly the a priori upper bound on the number of computational steps implies an upper bound on the number of times the non-deterministic choice can be made. From these two considerations it follows that the form of non-determinacy we have introduced is aptly described as "bounded non-determinacy". As a result also in the case of our bounded non-determinacy, we --or again at least the Good Lord-- could for each initial state decide a priori upon a sufficient size of a finite machine that could do the job.

The moral of the story is that a bounded number of computational steps and bounded non-determinacy together imply that the number of possible happenings and the maximum value possibly manipulated are both bounded as well.

The fact that the non-determinacy is bounded is very intimately tied to the guarded commands. For not only do they not introduce unbounded non-determinacy as we have seen, but also in the presence of unbounded non-determinacy --supplied by some other magical means-- the semantic

definition of the repetitive construct would be subject to doubt, to say the least.

Suppose that we have a non-deterministic primitive

"set x to any positive value"

and consider now the program

5: **do** $x > 0 \rightarrow x := x - 1$
$\quad [] \; x < 0 \rightarrow$ set x to any positive value **od**

I expect most readers to agree that, no matter what initial value of x, this program will terminate sooner or later with $x = 0$. If initially $x \geq 0$, then we know a priori the number of steps (viz. the initial value of x); if initially $x < 0$, we don't, but yet we have the feeling that it must terminate because then the second alternative will be chosen and after its successful execution, x will have some positive value that will be brought down to zero in a finite number of steps. As we shall see, the crux can be pinned down to the proviso "after its successful execution".

Our formalism, however, gives for $H_k(T)$ --the weakest pre-condition such that the construct terminates after at most k steps--:

$$H_k(T) = (0 \leq x \leq k)$$

and

$$\begin{aligned} wp(DO, T) &= (\underline{E}\ k: k \geq 0: H_k(T)) \\ &= (\underline{E}\ k: k \geq 0: 0 \leq x \leq k) \\ &= (0 \leq x) \quad . \end{aligned}$$

The inability to give for any negative x a priori an upper bound for

the number of steps needed is translated into "for negative x termination is not guaranteed". Have we failed?

That we have not can been seen by considering a possible implementation for "set x to any positive value", e.g.

```
go on:= true; x:= 1;
do go on → x:= x + 1
 [] go on → go on:= false
od
```

.

This construct will continue to increase x as long as the first alternative is chosen; as soon as the second alternative has been chosen once, it terminates immediately. Upon termination x may indeed be "any positive value" in the sense that we cannot think of a positive value X such that termination with $x = X$ is impossible. However, termination is not guaranteed either! If we substitute our implementation for "set x to any positive value" we get

```
do x > 0 → x:= x - 1
 [] x < 0 → go on:= true; x:= 1;
             do go on → x:= x + 1
              [] go on → go on:= false
             od
od
```

.

Now it is fully correct not to guarantee termination for initial states with $x < 0$, but not so much because we cannot say a priori how many steps the outer cycle will take, but because we cannot guarantee termination for the subprocess that would decide upon that number of steps: we cannot guarantee termination for the process that is expected to increase x to

an unbounded value! If it is expected to increase x possibly beyond any bound, we must also accept that it will go on for an arbitrarily long time, even forever.

There is still another reason for wanting to avoid unbounded non-determinacy. The repetitive construct gives rise to a weakest pre-condition of the form $(\underline{E}\ k:\ k \geq 0:\ H_k)$

where

for $k \geq 0$: $H_k \Rightarrow H_{k+1}$ for all states .

(This implication is intuitively clear from our interpretation "... after at most k executions of a guarded command"; formally it is easily proved by mathematical induction.) Such a condition might easily occur as post-condition for another statement, S say, and then we would like that

$$wp(S,(\underline{E}\ k:\ k \geq 0:\ H_k)) = (\underline{E}\ n:\ n \geq 0:\ wp(S,\ H_n)) \quad . \quad (1)$$

For a statement S with unbounded non-determinacy, however, this equality does not necessarily hold, as is shown by the example

H_k: $0 \leq x \leq k$

S: set x to any positive value .

Relation (1) does hold, however, under the assumption that the non-determinacy of S is bounded. In order to prove (1), we prove that for all states each side of (1) implies the other.

Consider an initial state such that the righthand side of (1) is true, i.e. there exists a value N such that in this initial state

$$wp(S,\ H_N)$$

holds. Because

$$H_N \Rightarrow (\underline{E}\ k:\ k \geq 0:\ H_k) \qquad \text{for all states}$$

we may conclude that

$$wp(S,\ H_N) \Rightarrow wp(S,\ (\underline{E}\ k:\ k \geq 0:\ H_k)) \qquad \text{for all states}$$

and therefore in the initial state considered the lefthand side of (1) must be true as well. We have proved that the righthand side of (1) implies the lefthand side of (1) in all states without an appeal on the bound on the non-determinacy; we need this bound to prove the implication the other way round.

Consider an initial state such that

$$wp(S,\ (\underline{E}\ k:\ k \geq 0:\ H_k)) \qquad ;$$

for such an initial state we guarantee termination in a bounded set of final states, each of them satisfying

$$(\underline{E}\ k:\ k \geq 0:\ H_k) \qquad ;$$

because each H_k implies all its successors, for each of these final states there is a minimum value kmin, such that

$$(\underline{A}\ k:\ k \geq kmin:\ H_k) \qquad .$$

Because the set of possible final states is bounded, the set of kmin-values is bounded and there is a maximum kmin-value, K say. As a result, our initial state satisfies

$$wp(S,\ H_K)$$

and as

$$wp(S,\ H_K) \Rightarrow (\underline{E}\ n:\ n \geq 0:\ wp(S,\ H_n))$$

the implication has been proved in the other direction as well. Relation (1) has been proved for all states.

As a consequence we can prove --with the usual meanings of BB, IF and DO -- that

$$(BB \text{ and } wp(DO, R)) = wp(IF, wp(DO, R)) \quad .$$

For $wp(IF, wp(DO, R)) = wp(IF, (\underline{E}\ k: k \geq 0: H_k(R)))$

$$= (\underline{E}\ n: n \geq 0: wp(IF, H_n(R)))$$

(and because $H_0(R) \Rightarrow$ non BB and for any P we have $wp(IF, P) \Rightarrow BB$)

$$= BB \text{ and } (\underline{E}\ n: n \geq 0: wp(IF, H_n(R)) \text{ or } H_0(R))$$

$$= BB \text{ and } (\underline{E}\ k: k \geq 1: H_k(R))$$

$$= BB \text{ and } wp(DO, R) \quad .$$

In other words, in all initial states such that BB holds

$$\underline{do}\ B_1 \rightarrow SL_1 \,[]\, B_2 \rightarrow SL_2 \,[]\, \ldots \,[]\, B_n \rightarrow SL_n\ \underline{od} \qquad (2)$$

is equivalent to

$$\underline{if}\ B_1 \rightarrow SL_1 \,[]\, B_2 \rightarrow SL_2 \,[]\, \ldots \,[]\, B_n \rightarrow SL_n\ \underline{fi};$$
$$\underline{do}\ B_1 \rightarrow SL_1 \,[]\, B_2 \rightarrow SL_2 \,[]\, \ldots \,[]\, B_n \rightarrow SL_n\ \underline{od} \quad . \qquad (3)$$

In initial states where BB does not hold, program (2) would have acted as "skip", while program (3) would have acted as "abort".

I find it somewhat disconcerting that so much has been involved in the formal proof of the partial equivalence of (2) and (3), the more so because this partial equivalence is intuitively so obvious. On the other hand it gives some confidence that it can be done. At the beginning of this chapter I have suggested that the majority of my readers should skip this chapter; at its end I express the hope that the few that have studied it will appreciate its inclusion.