Revised in EWD402

The characterization of semantics.

We are primarily interested in systems that, when started in an "initial state", will end up in a "final state" which, as a rule, depends on the choice of the initial state. This is a view that is somewhat different from the idea of the finite state automaton that on the one hand absorbs a stream of input characters and on the other hand produces a stream of output characters: to translate that in our picture we must assume that the value of the input (i.e. the argument) is reflected in the choice of the initial state and that the value of the output (i.e. the answer) is reflected in the final state. Our view relieves us from all sorts of peripheral complications.

Now I assume that the design of such a system is a goal-directed activity, in other words that we want to achieve something with the system. For instance, if we want to make a machine capable of computing the greatest common divisor, we could demand of the final state that it satisfies

$$x = \mathrm{GCD}(X,\ Y) \tag{1}$$

In the machine we have been envisaging, we shall then also have $y = \mathrm{GCD}(X,\ Y)$ --because the game terminates when $x = y$), but that is <u>not</u> part of our requirement when we decide to accept the final value of x as our "answer".

We call condition (1) the (desired) "post-condition" --"post" because it imposes a condition upon the state in which the system must find itself <u>after</u> its activity. Note, that the post-condition could be satisfied by many of the possible states. In that case we apparently regard each of them as equally satisfactory and there is then no reason to require that the final state is a unique function of the initial state.

In order to use such a system when we want it to produce an answer --say "reach a state satisfying condition (1) for a given set of values of X and Y-- we must know the set of corresponding initial states, more precisely: the set of initial states that is guaranteed to result in a final state satisfying (1). If we can bring the system without computational effort into one of these states, we know how to use the system to produce for us the desired answer! To give the example of Euclid's cardboard game: we can guarantee a final state satisfying the post-condition (1) for any initial state satisfying

$$GCD(x, y) = GCD(X, Y) \ \underline{\text{and}} \ \ 0 < x \le 500 \ \ \underline{\text{and}} \ \ 0 < y \le 500 \qquad (2).$$

(The upper limits have beed added to do justice to the limited size of the cardboard. If we start with a pair (X, Y) such that GCD(X, Y) = 713, then there exists no pair (x, y) satisfying (2), i.e. for those values of X and Y condition (2) reduces to F and that means that the machine in question cannot be used to compute the GCD(X, Y) for that pair of values of X and Y.

For many (X, Y)-combinations, many states satisfy (2). In the case that $0 < X \le 500$ and $0 < Y \le 500$, the <u>trivial</u> choice is $x = X$ and $y = Y$: it is a choice that can be made without any evaluation of the GCD-function, even without appealing to the fact that the GCD-function is a symmetric function of its arguments.

The condition that characterizes the set of <u>all</u> initial states that are guaranteed to lead to a final state satisfying a given post-condition is called "the weakest pre-condition corresponding to that post-condition".

(We call it "weakest", because the weaker a condition, the more states satisfy it and we aim here at characterizing all possible starting points that are certain to lead to one of the desired states.)

If the system (machine, mechanism) is denoted by "S" and the desired post-condition by "R", then we denote the corresponding weakest pre-condition by

$$wp(S, R) \quad .$$

If the initial state satisfies wp(S, R), the mechanism is certain to establish eventually the truth of R. Because wp(S, R) is the weakest pre-condition, we also know that if the initial state does not satisfy wp(S, R), the mechanism is not certain to establish eventually the truth of R. In the case that S is a deterministic machine --i.e. the final state is uniquely determined by the initial state-- it will then fail to do so, in the case that S is a non-deterministic machine, the most we can say is that then it may fail to do so.

As we proceed we shall see that there are a variety of ways in which the mechanism S may fail to reach a final state satisfying the post-condition R. One of the possibilities is that it reaches a final state for which R is false; another possibility is that it does not reach a final state at all, either because the system finds itself engaged in an endless task or because the system has got stuck.

We take the point of view that we know the possible performance of the mechanism S completely if and only if we can derive for any post-condition R the corresponding weakest pre-condition wp(S, R) because then we have captured completely what the mechanism can do for us. In the jargon: it is then that we know its "semantics".

Two remarks are in order. Firstly, the set of possible post-conditions is in general so huge, that this knowledge in tabular form --in a table with an entry for each R we would find the corresponding $wp(S, R)$-- would be utterly unmanageable, and therefore useless. Therefore the definition of the semantics of the mechanism is always given in another way, viz. in the form of a rule describing how for any given post-condition R the corresponding weakest pre-condition $wp(S, R)$ can be derived. Such a rule --which is fed with the predicate R denoting the post-condition and delivers a predicate $wp(S, R)$ denoting the corresponding weakest pre-condition-- is called "a predicate transformer". When we ask for the definition of the semantics of the mechanism S m what we really ask for is its corresponding predicate transformer.

Secondly, we are often --and I feel tempted to add "thank goodness"-- not interested in the complete semantics of a mechanism. This is because it is our intention to use the mechanism S for a specific purpose only, viz. for establishing the truth of a very specific post-condition R for which it has been designed. And even for that specific post-condition R , we are often not interested in the exact form of $wp(S, R)$: often we are content with a stronger condition P for which we can show that

$$P \Rightarrow wp(S, R) \qquad (3)$$

(read: "P implies $wp(S, R)$"). This means that wherever in the state space P is true, $wp(S, R)$ is true as well: P is a sufficient pre-condition. In terms of sets it means that the set of states characterized by P is a subset of the set of states characterized by $wp(S, R)$. If for a given

P, S and R relation (3) holds this can often be proved without explicit formulation --or if you prefer "computation" or "derivation"-- of the predicate wp(S, R). And that is a good thing for except in trivial cases we must expect that the explicit formulation of wp(S, R) will defy at least the size of our sheet of paper, our patience or our (analytical) ingenuity (or any combination of them).

The meaning of wp(S, R) : "the weakest pre-condition for the initial state such that the mechanism S will establish a final state satisfying the post-condition R " allows us to conclude that, considered as function of the post-condition R, the predicate transformer has a number of properties.

Property 0. For any mechanism S and any post-conditions R and Q, the equality "R = Q" alows us to conclude "wp(S, R) = wp(S, Q)" . In words: if R and Q are two predicates denoting the same post-condition, than the derived predicates wp(S, R) and wp(S, Q) denote the same pre-condition. One can argue whether it is worth-while to mention explicitly a property that is so obvious: if it did not hold, we had been talking nonsense all the time! I mention Property 0 for the sake of completeness. (I myself have worked for more than a month with predicate transformers without having formulated this property explicitly....)

Property 1. For any mechanism S we have "wp(S, F) = F" . Suppose that this was not true; under that assumption there would be at least one state satisfying wp(S, F) . Take such a state as the initial state for the mechanism S : then, according to our definitio of the weakest pre-condition "the mechanism S will establish a final state satisfying the post-condition F ", But this is a contradiction for there are by definition no states satisfying F . Property 1 is known under the name of "The Law of the Excluded Miracle".

<u>Property 2</u>. For any mechanism S and any post-conditions Q and R "Q => R" allows us to conclude "wp(S, Q) => wp(S, R)". Indeed, because any initial state satisfying wp(S, Q) is "such that the mechanism S will establish a final state satisfying the post-condition Q"; on account of "Q => R" any such final state will satisfy R as well, i.e. any initial state satisfying wp(S, Q) is such that the mechanism will establish a final state satisfying the post-condition R and therefore any initial state satisfying wp(S, Q) will satisfy wp(S, R) as well. Property 2 is called "The property of Monotonicity".

<u>Property 3</u>. For any mechanism S and any post-conditions Q and R

$$(\mathrm{wp}(S, Q)\ \underline{\text{and}}\ \mathrm{wp}(S, R)) = \mathrm{wp}(S, Q\ \underline{\text{and}}\ R) \qquad (4).$$

The lefthand side of (4) implies the righthand side because for any initial state satisfying both wp(S, Q) and wp(S, R) we have the combined knowledge that a final state will be established satisfying both Q and R. Also, on account of the Property of Monotonicity we conclude from (Q <u>and</u> R) => Q that wp(S, Q <u>and</u> R) => wp(S, Q) ; similarly we conclude that wp(S, Q <u>and</u> R) => wp(S, R) and from the last two implications we conclude that the righthand side of (4) implies the lefthand side. Both sides implying eachother, they must be equal and thus Property 3 has been proved.

<u>Property 4</u>. For any mechanism S and any post-condition Q and R

$$(\mathrm{wp}(S, Q)\ \underline{\text{or}}\ \mathrm{wp}(S, R)) \Rightarrow \mathrm{wp}(S, Q\ \underline{\text{or}}\ R) \qquad (5)$$

Also this is a direct consequence of the Property of Monotonicity. From $Q \Rightarrow Q\ \underline{\text{or}}\ R$ follows $wp(S, Q) \Rightarrow wp(S, Q\ \underline{\text{or}}\ R)$; from $R \Rightarrow Q\ \underline{\text{or}}\ R$ follows $wp(S, R) \Rightarrow wp(S, Q\ \underline{\text{or}}\ R)$. And then, relation (5) follows.

We can make a stronger assertion for the special case that the mechanism is deterministic. The deterministic machine has the special property that its behaviour --i.e. if a final state will be reached and, if so, which one-- is fully determined by the initial state. While in general we can only be sure that the final state will satisfy the post-condition R provided that the initial state satisfies $wp(S, R)$, we know for the deterministic machine S that the final state will satisfy the post-condition R if <u>and only if</u> the initial state satisfies $wp(S, R)$: this is because every initial state that could lead to a final state satisfying R <u>must</u> lead to that final state satisfying R . In the special case of the deterministic machine the righthand side of (5) implies therefore the lefthand side as well and we have

$$(wp(S, R)\ \underline{\text{or}}\ wp(S, Q)) = wp(S, R\ \underline{\text{or}}\ Q) \quad . \qquad (6)$$

In this book --and that may turn out to be its distinctive feature-- I shall treat non-determinacy as the rule and determinacy as the exception: a deterministic machine will be regarded as a special case of the non-deterministic one, as a mechanism for which Property 4 is given by (6) rather than by the somewhat weaker (5). This decision reflects a drastic change in my own thinking. Back in 1958 I have been one of the firsts to develop the basic software for a machine with an I/O interrupt and the irreproducibility of the behaviour of such a --to all intents and purposes: non-deterministic-- machine has been a traumatic experience. When the idea of the I/O interrupt was first suggested Iwas so terrified at the thought of having to build reliable siftware for such an intractable beast that I have delayed the decision to incorporate the feature for at least three months. And even after I had given in --I had been flattered out of my resistance!-- I was highly uncomfartable. When the prototype was becoming king of operational I had my worst fears fully confirmed: a bug in the program could evoke the erratic behaviour so strongly suggestive of an irreproducible machine error. And secondly --and that was in the time that for deterministic machines we still believed in "debugging"-- it was right from the start quite obvious that program testing was quite ineffective as a means for rasing the confidence level.

For many years thereafter I have regarded the irreproducibility of the behaviour of the non-deterministic machine as an added complication that should be avoided whenever possible. Interrupts were nothing but a curse inflicted by the hardware engineers upon the poor software makers. Out of this fear of mine the discipline for "harmoniously co-operating sequential processes" has been born. In spite of its success I was still afraid for our solutions --although proved to be correct-- seemed ad hoc solutions to the problem of "taming" --that is the way we felt about it!-- special forms of non-determinacy. The background of my fear was the absence of a general methodology.

Two circumstances have changed the scene since then. The one is the discovery that, even in the case of fully deterministic machines, program testing is hardly helpful. As I have now said many times and written in many places: program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence. The other one is the discovery that in the mean time it has emerged that any design discipline must do justice to the fact that the design of a mechanism that is to have a purpose, must be a goal-directed activity. In our special case it means that we can expect our post-condition to tbe the starting point of our design considerations. In a sense we shall be "working backwards". In doing os we shall find that the implication of relation (5) is the essential part, for the equality of relation (6) we shall have very little use.

Once the mathematical equipment needed for the design of non-deterministic mechanisms achieving a purpose has been developed, the non-deterministic machine is no longer frightening, on the contrary! We shall learn to appreciate it even as a valuable steeping stone in the design of an ultimately fully deterministic mechanism.

*　　　*　　　*

Above we have mentioned that the complete evaluation of $wp(S, R)$ is often beyond our interest and/or power and that we are content with a sufficient pre-condition such that $P \Rightarrow wp(S, R)$, i.e. that P is a sufficient pre-condition to guarantee that the mechanism S will reach a final state satisfying R .

A closely related but subtly different concept is that of the "safe pre-condition". We call "P with respect to the mechanism S a safe pre-condition for the post-condition R if an initial state satisfying P guarantees that the mechanism S cannot reach a final state not satisfying R ". In formula we denote this state of affairs by

$$\{P\}\ S\ \{R\}\quad .$$

Note that the two negations in the end of the definition of the concept of "a safe pre-condition" do not cancel: an initial state satisfies a safe pre-condition either because it will lead to a final state satisfying R or to no state at all (i.e. when the mechanism S fails to terminate properly) or both.

The concept of a safe pre-condition derives its usefulness from the following property. From

$$\{P\}\ S\ \{R\}$$

follows $$(wp(S, T)\ \underline{\text{and}}\ P) \Rightarrow wp(S, R) \qquad . (7)$$

Here the term $wp(S, T)$ describes the weakest pre-condition such that S will reach "a final state satisfying T ", but as each state satisfies T by definition, this reduces to "the weakest pre-condition such that S will reach a final state" or "will terminate properly". From this observation and the definitions, the above property follows immediately. It tells us that "$(wp(S, T)\ \underline{\text{and}}\ P)$" is a sufficient pre-condition for the establishment of the truth of R .

The above relation between safe and sufficient pre-conditions will play a central role in the practice of program composition. The point is that termination is in general a tricky problem, but in deriving a safe pre-condition for the post-condition R we can ignore the termination problem. When establishing the termination we can ignore the post-condition R ! In formula (7) the requirements

1) that the mechanism will terminate, and

2) that the final state will satisfy R

have been factored nicely: two different concerns have been separated.