On the axiomatic definition of semantics

(The following is heavily influenced by the work of C.A.R.Hoare.)

Program testing can be used very effectively to show the presence of bugs, but is hopelessly inadequate for showing their absence and a convincing correctness proof seems the only way to reach the required confidence level.

In order that such a convincing correctness proof may exist, two conditions must be satisfied by such a correctness proof:
1) it must be a proof and that implies that we need a set of axioms to start with
2) it must be convincing and that implies that we must be able to write, to check, to understand and to appreciate the proof.
This essay deals with the first of these two topics.

We are considering finite computations only; therefore we can restrict ourselves to computational processes taking place in a finite state machine -although the possible number of states may be very, very large- and take the point of view that the net effect of the computation can be described by the transition from initial to final state. The computation is assumed to take place under control of an algorithm, a program, and what we want to do is to make assertions about all possible computations that may be evoked under control of such a program. And we want to base these assertions on the program text!

This implies that we must have a formal definition of the semantics of the programming language in which the program has been expressed.

The earliest efforts directed towards such definition of semantics that I am aware of have been what I call "mechanistic definitions": they gave a definition (or "a description") of the steps that should be carried out in executing a program, they gave you "the rules of the game", necessary to carry out any given computation (as determined by program and initial state!) by hand (or by machine). The basic shortcoming of this approach was that the semantics of an algorithm were expressed in terms of "the rules of the game", i.e. in terms of another algorithm. The game can only be played for a chosen

initial state, and as a result it is as powerless as program testing! A mechanistic definition as such is not a sound basis for making assertions about the whole class of possible computations associated with a program. It is this shortcoming, that the axiomatic method seeks to remedy.

We consider predicates P, Q, R, ... on the set of states; for each possible state a given predicate will be either true of false and if we so desire, we can regard the predicate as characterizing the subset of states for which it is true. There are two special predicates, named T and F: T is true for all possible states (characterizes the universe), F is false for all possible states (characterizes the empty set). We call two predicates P and Q equal ("P = Q") when the sets of states for which they are true are the same. (Note that P ≠ T -or non(P = T)- does not allow us to conclude P = F !)

We consider the semantics of a program S fully determinedm when we can derive for any set of final states, as characterized by a predicate P (called in this connection "the poast-condition") the complete set of corresponding initial states, a set characterizes by the predicate that we denote by fS(P) and that in this connection is called "the weakest pre-condition" or "the pre-condition" for short. Here we regard the fS as a "predicate transformer", as a rule for deriving the weakest pre-condition from the post-condition to which it corresponds.

The semantics of a program S are defined when the corresponding predicate transformer fS is given, the semantics of a programming language are defined when the rules are given how to construct the predicate transformer fS corresponding to a program S written in that language.

As most programming languages are defined recursively, we can expect such construction rules for the predicate transformer of the total program to be expressed in terms of predicate transformers associated with components. But, as we shall see in a moment, we must observe some restrictions, for if we allow ourselves too much freedom in the construction of predicate transformers we may arrive at predicate transformers fS such that fS(P) can no longer be interpreted as the weakest pre-condition corresponding to the post-condition P for a possible deterministic machine.

Our construction rules for predicate transformers fS must be such that, whatever fS we construct, it must have the following four basic properties

1) $P = Q$ implies $fS(P) = fS(Q)$

2) $fS(F) = F$

3) $fS(P \text{ and } Q) = fS(P) \text{ and } fS(Q)$

4) $fS(P \text{ or } Q) = fS(P) \text{ or } fS(Q)$

Predicate transformers enjoying those four properties we call "healthy".

Property 1 assures that we are justified in regarding the predicates as characterizing our true subject matter, viz. sets of states: it would be awkward if $fS(x > 0)$ would differ from $fS(0 < x)$!

Property 2 is the so-called "Law of the Excluded Miracle" and does not need any further justification.

The justification for properties 3 and 4 becomes fairly obvious when we consider, for instance $P = (0 \leq x \leq 2)$ and $Q = (1 \leq x \leq 3)$ and require that each initial state satisfying $fS(P)$ is mapped into a single state satisfying P and similarly for Q. Conversely it can be shown that each healthy predicate transformer fS can be interpreted as describing the net effect of a deterministic machine, whose actions are fully determined by the initial state.

From our 1st and 4th property we can derive a conclusion. Let $P \Rightarrow Q$; from this is follows that there exists a predicate R such that we can write $Q = P \text{ or } R$; our 1st and 4th property then tell us that

$$fS(Q) = fS(P \text{ or } R) = fS(P) \text{ or } fS(R)$$

from which we deduce that

5) $P \Rightarrow Q$ implies $fS(P) \Rightarrow fS(Q)$.

The simplest predicate transformer enjoying the four basic properties is the identity transformation:

$$fS(P) = P \quad ;$$

the corresponding statement is well known to programmers, they usually call it "the empty statement".

But it is very hard to build up very powerful programs from empty statements alone, we need something more powerful. We really want to transform a given predicate P into a possibly different predicate fS(P).

One of the most basic operations that can be performed upon formal expressions is substitution, i.e. replacing all occurrences of a variable by (the same) "something else". If in the predicate P all occurrences of the variable "x" are replaced by (E), then we denote the result of this transformation by

$$P_{E \rightarrow x} \quad .$$

Now we can consider statements S such that

$$fS(P) = P_{E \rightarrow x} \quad ;$$

this is a whole class of statements, they are given by three things

a) the identity of the variable x to be replaced

b) the fact that the substitution is the corresponding rule for predicate transformation

c) the expression E which is to replace every occurrence of x in P.

The usual way to write such a statement is

$$x := E$$

and such a statement is known under the name of an "assignment statement". We can formulate the

Axiom of Assignment. When the statement S is of the form $x := E$, then its semantics are given by the predicate transformer fS that is such that for all P $\quad fS(P) = P_{E \rightarrow x}$.

The substitution process leads to predicate transformers that are healthy.

Although from a logical point of view unnecessary -we can take this predicate transformer to give by definition the semantics of what we call assignment statements- it is wise to confront this axiomatic definition with our intuitive understanding of the assignment statement -if we have one!- and it is comforting to discover that indeed it captures the assignment statement as we (may) know it, as the following examples -written in the format: $\{fS(P)\}\ S\ \{P\}$- show:

$$\{\ a > 0\ \}\ x := 1\ \{\ a > 0\}$$

$$\{\ (1) < 2\ \}\ x := 1\ \{\ x < 2\ \}$$

$$\{\ a > 0 \text{ and } (x + 1) < 9\ \}\ x := x + 1\ \{\ a > 0 \text{ and } x < 9\ \} \quad .$$

The above rules enable us to establish the semantics of the empty program and of the program consisting of a single assignment statement. In order to be able to compose more complicated predicate transformers, we observe that the functional composition of two healthy predicate transformers is again healthy. So this is a legitimate way of constructing a new one and we are lead to the

Axiom of Concatenation. Given two statements S1 and S2 with (healthy) predicate transformers fS1 and fS2 respectively, the predicate transformer fS, given for all P by

$$fS(P) = fS1(fS2(P))$$

is healthy and taken as the semantic definition of the statement S that we denote by S1 ; S2 .

Functional composition is associative and we are therefore justified in the use of the term "concatenation": it makes no difference if we parse "S1 ; S2 ; S3" either as "(S1 ; S2); S3" or as "S1 ;(S2 ; S3)".

Relating the axiomatic definition of the concatenation operator ";" to our intuitive understanding of a sequential computation, it just means that each execution of S1 (when completed) will immediately be followed by an execution of S2 and, conversely, that each execution of S2 has immediately been preceded by an execution of S1. The functional composition identifies the initial state of S2 with the final state of S1.

Looking for new programming language constructs implies looking for new ways of constructing predicate transformers, but all this, of course, subject to the restriction that the ensuing predicate transformer must be healthy. And a number of obvious suggestions must be rejected on that grounds, such as: $fS(P) = \underline{non}\ fS1(P)$
for that would violate the Law of the Excluded Miracle.
Also $fS(P) = fS1(P)\ \underline{and}\ fS2(P)$
must be rejected as such a fS violates the basic property 4:

$$fS(P\ \underline{or}\ Q) = fS1(P\ \underline{or}\ Q)\ \underline{and}\ fS2(P\ \underline{or}\ Q) = \{fS1(P)\ \underline{or}\ fS1(Q)\}\ \underline{and}\ \{fS2(P)\ \underline{or}\ fS2(Q)\}$$

while

$$fS(P)\ \underline{or}\ fS(Q) = \{fS1(P)\ \underline{and}\ fS2(P)\}\ \underline{or}\ \{fS1(Q)\ \underline{and}\ fS2(Q)\}$$

and they are in general different, as the first of the two leads to the additional terms in the disjucntion

$$\{fS1(P) \text{ and } fS2(Q)\} \text{ or } \{fS1(Q) \text{ and } fS2(P)\}.$$

Similarly, if we choose

$$fS(P) = fS1(P) \text{ or } fS2(P)$$

property 3 is violated, because

$$fS(P \text{ and } Q) = fS1(P \text{ and } Q) \text{ or } fS2(P \text{ and } Q) = \{fS1(P) \text{ and } fS1(Q)\} \text{ or } \{fS2(P) \text{ and } fS2(Q)\}$$

while

$$fS(P) \text{ and } fS(Q) = \{fS1(P) \text{ or } fS2(P)\} \text{ and } \{fS1(Q) \text{ or } fS2(Q)\}$$

and here the second one leads to the additional terms in the disjunction

$$\{fS1(P) \text{ and } fS2(Q)\} \text{ or } \{fS1(Q) \text{ and } fS2(P)\} .$$

This leads to the suggestion that we look for fS1 and fS2 (in general fS_i) such that for any P and Q

$$i \neq j \text{ implies } \quad fS_i(P) \text{ and } fS_j(Q) = F.$$

Doing it only for a pair leads to the

Axiom of Binary Selection. Given two statements S1 and S2 with healthy predicate transformers fS1 and fS2 respectively and a (computable) predicate B, the predicate transformer fS, given for all P by

$$fS(P) = \{B \text{ and } fS1(P))\} \text{ or } (\text{non } B \text{ and } fS2(P)\}$$

is healthy and taken as the semantic definition of the statement S that we denote by

$$\text{if } B \text{ then } S1 \text{ else } S2 \text{ fi} \quad .$$

(This is readily extended to a choice between three, four or any explicitly enumerated set of mutually exclusive alternatives, leading to the so-called case-construction.)

For an arbitrary given sequence fS_i we can not hope, that $i \neq j$ implies $fS_i(P) \text{ and } fS_j(Q) = F$ for any P and Q, but we may hope to achieve this if we can generate the fS_i by a recurrence relation. Before we embark upon such a project, however, we should derive a useful property of the predicate transformers we have been willing to construct thus far.

If two predicate transformers fS and fS' satisfy the property that for all P: $fS(P) \Rightarrow fS'(P)$, then we call "fS as strong as fS'" and "fS' as weak as fS".

(The predicate transformer given for all P by $fS(P) = F$ is as strong as any other, the predicate transformer given by $fS(P) = T$ would be as weak as any other if it were admitted, but it is not, because it is not healthy: it violates the Law of the Excluded Miracle.)

We can now formulate and derive our

Theorem of Monotony. Whenever in a predicate transformer fS, formed by concatenation and/or selection, one of the constituent predicate transformers is replaced by one as weak (strong) as the original one, the resulting predicate transformer fS' is as weak (strong) as fS.

Obviously we only need to show this for the elementary transformer constructions.

Concatenation, case 1:

Let S be: S1 ; S2

let S' be: S1'; S2

let S1' be as weak as S1, then

$fS(P) = fS1(Q)$ and $fS'(P) = fS1'(Q)$, with $Q = fS2(P)$; as $fS1(Q) \Rightarrow fS1'(Q)$ for any Q, $fS(P) \Rightarrow fS'(P)$ for any P. QED.

Concatenation, case 2:

Let S be: S1 ; S2

let S' be: S1 ; S2'

let S2' be as weak as S2, then

$fS(P) = fS1(Q)$ and $fS'(P) = fS1(R)$ where $Q = fS2(P)$ and $R = fS2'(P)$. Because for any P, $Q \Rightarrow R$, it follows from the healthiness of fS1, that $fS(P) \Rightarrow fS'(P)$. QED.

Binary selection, case 1:

Let S be: if B then S1 else S2 fi

let S' be: if B then S1' else S2 fi

let S1' be as weak as S1, then

$fS(P) = \{B \text{ and } fS1(P)\} \text{ or } \{\text{non } B \text{ and } fS2(P)\}$

$\Rightarrow \{B \text{ and } fS1'(P)\} \text{ or } \{\text{non } B \text{ and } fS2(P)\} = fS'(P)$. QED.

Binary selection, case 2 can be left to the industrious reader.

Let us now consider a predicate transformer G constructed by means of concatenation and selection, out of a number of healthy predicate transformers, among which fH. (This predicate transformer may be used "more than once": then it corresponds to a program text in which the corresponding statement H occurs more than once.) We wish to regard this predicate transformer as a function of fH and indicate that by writing G(fH), i.e. G derives by concatenation and/or selection with other, in this connection fixed predicate transformers, a new predicate transformer. We now consider the recurrence relation

$$fH_i = G(fH_{i-1}) \qquad (1)$$

which is a tractable thing in the sense that if fH_0 is as strong (weak) as fH_1, it follws via mathematical induction from the Theorem of Monotony that fH_i is as strong (weak) as fH_{i+1} for all i. We should like to start the recurrence relation with a constant transformer fH_0, that is either as strong or as weak as any other. We can do this for a predicate transformer as strong as any other by choosing $fH_0 = fSTOP$, given by

$$fSTOP(P) = F \text{ for any } P \quad .$$

(The predicate transformer fSTOP satisfies all the requirements for healthiness.) And so we find ourselves considering the sequence of predicate transformers, given by $fH_0 = fSTOP$

and for $i > 0$: $fH_i = G(fH_{i-1})$ (2)

with the property that

1) all fH_i are healthy (by induction)

2) for $i \leq j$ and any P: $fH_i(P) \Rightarrow fH_j(P)$.

Because all fH_i are healthy and any $P \Rightarrow T$, we also know that for any P

$$fH_i(P) \Rightarrow fH_i(T) \quad .$$

We now recall that we were looking for fS_i such that for any P and Q and $i \neq j$ we would have $fS_i(P) \text{ and } fS_j(Q) = F$.

We can derive such predicate transformers from the $fH_i(P)$. As each $fH_i(P)$ implies the next one in the sequence we could try for $i \geq 0$

$$fS_i(P) = fH_i(P) \text{ and non } fH_{i-1}(P)$$

i.e. the $fS_i(P)$ is the "incremental tolerance", but -both on account of the conjunction and on account of the negation- it is not obvious that such a construction is a healthy predicate transformer.

Therefore we proceed a little bit more carefully, first deriving a result that is independent of the post-condition. Each $fH_i(P)$ implies the next one in the sequence for any P and therefore also for P = T. We now define

$$K_i = fH_i(T) \text{ and non } fH_{i-1}(T) \text{ for } i > 0 \tag{3}$$

and derive about the K_i the following properties. (If we so desire, we can define $K_0 = F$.)

$$1) \quad i \neq j \text{ implies } K_i \text{ and } K_j = F \tag{4}$$

This is proved by a reduction ad absurdum. Let $i < j$ and suppose K_i and $K_j \neq F$; then there exists a point x in state space such that

$$K_i(x) \text{ and } K_j(x) = \text{true} \quad .$$

However, $K_i(x)$ implies $fH_i(T)(x)$ which implies $fH_{j-1}(T)(x)$ -because $j-1 \geq i$ - which implies $K_j(x) = \text{false}$ and this is the contradiction we were after. In words: in each point in state space at most one K_i is true.

$$2) \quad fH_i(T) = (\underline{E}\ j:\ 1 \leq j \leq i:\ K_j) \tag{5}$$

From $fH_0(T) = F$, we derive $K_1 = fH_1(T)$ so (5) holds for i = 1. For the induction step we observe that

$fH_i(T) =$
$fH_i(T) \text{ and } (\text{non } fH_{i-1}(T) \text{ or } fH_{i-1}(T)) =$
$K_i \text{ or } \{fH_i(T) \text{ and } fH_{i-1}(T)\} = K_i \text{ or } fH_{i-1}(T)$.

The healthiness of fH_i allows another conclusion.

$$fH_i(P) \text{ or } fH_i(\text{non } P) = fH_i(P \text{ or non } P) = fH_i(T) \tag{6}$$
$$fH_i(P) \text{ and } fH_i(\text{non } P) = fH_i(P \text{ and non } P) = fH_i(F) = F \tag{7}$$

Taking in (6) at both sides the conjunction with non $fH_i(P)$ we reach

$fH_i(\text{non } P) \text{ and non } fH_i(P) = fH_i(T) \text{ and non } fH_i(P)$;

taking in the last two formulae at both sides the disjunction, we find

$$fH_i(\text{non } P) = fH_i(T) \text{ and non } fH_i(P)$$

and as $fH_i(\text{non } P) \Rightarrow fH_j(\text{non } P)$ for $j \geq i$
we conclude for $j \geq i$

$$fH_i(T) \text{ and non } fH_i(P) \Rightarrow fH_j(T) \text{ and non } fH_j(P)$$

or

$$K_i \text{ and non } fH_i(P) \Rightarrow \text{non } fH_j(P) \tag{8}$$

Now, at last, we are ready to prove the theorem I have been driving at all the time, viz.

$$fH_i(P) = (\underline{E}j: 1 \leq j \leq i: K_j \ \underline{and}\ fH_j(P)) \qquad (9)$$

In one direction the proof is easy. Consider a point x in state space, such that $fH_i(P)(x) = \underline{false}$; on account of the monotomy of the fH_i we can conclude that then $j \leq i$ implies $fH_j(P)(x) = \underline{false}$, therefore the right-hand side is $\underline{false}$ as well.

Consider now a point x, such that $fH_i(P)(x) = \underline{true}$. Then $fH_i(T)(x) = \underline{true}$ and according to (4) and (5) there is exactly one value m in the range $1 \leq m \leq i$ such that $K_m(x) = \underline{true}$. Furthermore, on account of the monotony of the fH_i, there is a unique value k in the range $1 \leq k \leq i$ such that

$j \geq k$ implies $fH_j(P)(x) = \underline{true}$

$j < k$ implies $fH_j(P)(x) = \underline{false}$.

Now relation (8) excludes the case $m < k$, for then we would have

$$K_m(x) \ \underline{and}\ \underline{non}\ fH_m(P)(x) = \underline{true}$$

implying (as $i \geq m$) $\underline{non}\ fH_i(P)(x)$, contradicting our assumption. Therefore $m \geq k$, implying $K_m \ \underline{and}\ fH_m(P)(x) = \underline{true}$, which leads to the conclusion that the right-hand side is true as well. And thus relation (9) has been proved.

And now most of the hairy work has been done. Because fH_i is a healthy predicate transformer, fS_i, given by

$$fS_i(P) = K_i \ \underline{and}\ fH_i(P) \qquad (10)$$

is healthy as well. (Because K_i is a predicate independent of P, it is the predicate transformer associated with

$$\underline{if}\ K_i\ \underline{then}\ H_i\ \underline{else}\ \text{STOP}\ \underline{fi}\ .)$$

But on account of (4) we now have: $i \neq j$ implies $fS_i(P) \ \underline{and}\ fS_j(Q) = F$ (11)

and this false conjunction was exactly what we were looking for! With the aid of our infinite sequence of fS_i we can now form two new healthy predicate transformers, firstly

$$fH(P) = (\underline{A}\ i: 1 \leq i: fS_i(P)) \quad ,$$

but that one, although healthy, is not interesting because on account of (11) it is identically F, and secondly

$$fH(P) = (\underline{E}\ i:\ 1 \leq i:\ fS_i(P)) \quad . \tag{12}$$

this one is <u>not</u> identically F and we call it a predicate transformer "composed by recursion". In formula (12), for each point x in state space, such that $fH(P)(x) = \underline{true}$, the existential quantifier singles out a unique value of i.

Alternatively we may write

$$\begin{aligned} fH(P) &= (\underline{E}\ j:\ 1 \leq j:(\underline{E}\ i:\ 1 \leq i \leq j:\ fS_i(P))) \\ &= (\underline{E}\ j:\ 1 \leq j:\ fH_j(P)) \quad . \end{aligned} \tag{13}$$

As formula (9) is equivalent to

$$fH_i(P) = (\underline{E}\ j:\ 1 \leq j \leq i:\ fS_i(P))$$

we see, on account of (11) that, as we guessed

$$fS_i(P) = fH_i(P)\ \underline{and}\ \underline{non}\ fH_{i-1}(P)$$

but in the mean time its healthyness has been established.

* * *

I apologize most sincerely for the last few pages of formal labour, which took me a few days to write down , while a greater expert in the propositional calculus probably could have done it much more efficiently. It is by now most urgent that we relate the above to our intuitive understanding of the recursive procedure: then all our formulae become quite obvious (and the painfulness of my formal proofs becomes frustrating!)

First a remark about the Theorem of Monotony: it just states that if we replace a component of a structure by a more powerful one, the modified structure will be at least as powerful as the original one. (Consider, for instance, an implementation of a programming language that leads to program abortion when integer overflow occurs, i.e. when an integer value outside the range $[-M, +M]$ is generated.When we modify the machine by increasing M, all computations that were originally feasible, remain so, but possibly we can do more.)

Now for the recursion. All we have been talking about is a recursive procedure (without local variables and without parameters) that could have been declared by a text of the form

proc H: .....H.....H.....H..... corp

i.e. a procedure H that may call itself from various places in its body. Mentally we are considering an infinite sequence of procedures H_i with

proc H_0: STOP corp

proc H_i: .....H_{i-1}.....H_{i-1}.....H_{i-1}..... corp .

Our rules $fH_0 = fSTOP$ and for $i > 0$: $fH_i = G(fH_{i-1})$

are such that the proposition transformer fH_i corresponds to our intuitive understanding of the call of procedure H_i. In terms of the procedure H, fH_i describes what a call of the procedure H can do under the additional constraint that the dynamic recursion depth will not exceed i. In particular, $fH_i(T)$ characterizes the initial states such that the procedure call will terminate with a dynamic recursion depth not exceeding i, while K_i characterizes those initial states such that a call of H will give rise to a maximum recursion depth exactly = i. This intuitive interpretation makes our earlier formulae quite obvious, fH(T) is the weakest precondition that the call will lead to a terminating computation.

The Theorem of Monotony was proved for predicate transformers formed by concatenation and/or selection. If in the body of H one of the predicate transformers fS is replaced by fS', as weak(strong) as fS, then G'(fH) will be as weak (strong) as G(fH), giving rise to an fH'_i as weak (strong) as fH_i; as a result the Theorem of Monotony holds also for predicate transformers constructed via recursion.

Our axiomatic definition of the semantics of a recursive procedure

$$fH_0 = fSTOP$$

and for $i \geq 0$: $fH_i = G(fH_{i-1})$

and $$fH(P) = (\underline{E}\ i:\ i \geq 1:\ fH_i(P)) \qquad (14)$$

is nice and compact, in actual practice it has one tremendous disadvantage: for all but the simplest bodies, it is impossible to use it directly. $fH_1(P)$ becomes a line, $fH_2(P)$ becomes a page, etc. and this circumstance makes it often very unattractive to use it directly. We cannot blame our axiomatic definition of the recursive procedure for this unattractive state of affairs: recursion is such a powerful technique for the construction of new predicate transformers, that we can hardly expect a recursive procedure "chosen at random" to turn out to be a mathematically manageable object. So we had better

discover which recursive procedures can be managed intellectually and how. This is nothing more nor less than asking for useful theorems about the semantics of recursive procedures.

Ou intuitive understanding of the K_i -i.e. the initial states, such that a call will lead to a maximum recursion depth = i- suggests a theorem: if $i > 1$, it can only achieve its maximum recursion depth by primarily generating at least one call that gives rise to a maximum recursion depth = i-1. Therfore we conjecture

$$\text{for } 1 \leq i \leq j \ : \quad K_i = F \text{ implies } K_j = F \quad . \tag{15}$$

On account of the assumption

$$K_i = fH_i(T) \ \underline{\text{and}} \ \underline{\text{non}} \ fH_{i-1}(T) = F$$

we have

$$T = \underline{\text{non}} \ K_i = \{\underline{\text{non}} \ fH_i(T) \ \underline{\text{or}} \ fH_{i-1}(T)\} = \{fH_i(T) \Rightarrow fH_{i-1}(T)\};$$

on the other hand we have $fH_{i-1}(T) \Rightarrow fH_i(T)$ and therefore $fH_i(T) = fH_{i-1}(T)$. We would like to conclude

$$fH_i = fH_{i-1} \tag{16}$$

or, more explicitly,

$$fH_i(P) = fH_{i-1}(P) \text{ for any } P \quad . \tag{17}$$

The latter relation is true, if it is true for any point x in state space. If $fH_i(T)(x) = \underline{\text{false}}$, so are $fH_i(P)(x)$ and $fH_{i-1}(P)(x)$, and we are left with the points x, such that $fH_i(T)(x) = \underline{\text{true}}$.

Because fH_i is healthy we have for any P

$$fH_i(P)(x) \ \underline{\text{or}} \ fH_i(\underline{\text{non}} \ P)(x) = fH_i(T)(x) = \underline{\text{true}}$$

$$fH_i(P)(x) \ \underline{\text{and}} \ fH_i(\underline{\text{non}} \ P)(x) = fH_i(F)(x) = \underline{\text{false}} \quad ,$$

from which it follows that

$$fH_i(P)(x) = \underline{\text{non}} \ fH_i(\underline{\text{non}} \ P)(x)$$

and similarly for the same arbitrary P and the same point x

$$fH_{i-1}(P)(x) = \underline{\text{non}} \ fH_{i-1}(\underline{\text{non}} \ P)(x) \quad .$$

If $fH_{i-1}(P)(x) = \underline{\text{true}}$, we see that $fH_i(P)(x) = \underline{\text{true}}$, because $fH_{i-1}(P) \Rightarrow fH_i(P)$;

if $fH_{i-1}(P)(x) = \underline{\text{false}}$, $fH_{i-1}(\underline{\text{non}}\ P)(x) = \underline{\text{true}}$ and so is $fH_i(\underline{\text{non}}\ P)(x)$, because $fH_{i-1}(\underline{\text{non}}\ P) \Rightarrow fH_i(\underline{\text{non}}\ P)$, and thus $fH_i(P)(x) = \underline{\text{false}}$ as well.

Therefore (17) and (16) as well have been established. But on account of the recurrence relation for the fH_i

$$fH_i = fH_{i-1} \quad \text{implies} \quad fH_{i+1} = fH_i$$

and via mathematical induction $fH_j = fH_{j-1}$ for any $j \geq i$, in particular $fH_j(T) = fH_{j-1}(T)$ and (15) has been established. (Whether in the sequel we can make good use of (15) is still an open question; for the time being we can regard its proof as an exercise.)

* * *

Now we are going to prove the Fundamental Invariance Theorem for Recursive Procedures.

Consider a text, xalled H", of the form

H":.....H'.....H'.....H'.....

to which corresponds a predicate transformer fH", such that for two specific predicates Q and R, the assumption $Q \Rightarrow fH'(R)$ is a sufficient assumption about fH' to prove $Q \Rightarrow fH"(R)$.

In that case, the recursive procedure H given by

proc H: .....H.....H.....H.....corp

(where we get this text by removing the dashes and enclosing the resulting text between the brackets proc and corp) enjoys the property that

$$Q \ \underline{\text{and}}\ fH(T) \Rightarrow fH(R) \quad . \tag{18}$$

(The tentatice conclusion $Q \Rightarrow fH(R)$ is wrong, as is shown by the counter-example proc H: H corp .)

We show this by showing that then Q must be such that for all i

$$Q \ \underline{\text{and}}\ fH_i(T) \Rightarrow fH_i(R) \tag{19}$$

and from (19), (18) follows trivially.

Relation (19) holds for i = 0, mathematical induction is the tool and we propose to demonstrate that if (19) holds for i = j-1, it will hold for i = j as well.

Let us consider first, for the sake of simplicity, the case that the text H22 contains a single reference to H'. In the evaluation of fH"(R), let P1 be the argument supplied to fH'; with

$$P2 = fH(P1)$$

we can write $fH"(R) = E(P2)$.

We can regard E as a predicate transformer operating on its argument P2, but considered as predicate transformer it is not necessarily healthy: it may violate the Law of the Excluded Miracle. It enjoys, however, the other three properties:

$P = Q$ implies $E(P) = E(Q)$

$E(P \text{ and } Q) = E(P) \text{ and } E(Q)$

$E(P \text{ or } Q) = E(P) \text{ or } E(Q)$

and therefore also the fifth:

$P \Rightarrow Q$ implies $E(P) \Rightarrow E(Q)$.

The statement that the conditions A and B are such that the assumption $A \Rightarrow fH'(R)$ is a sufficient assumption about fH', such that we can prove $B \Rightarrow fH"(R)$ is, in terms of P1 and E, equivalent to the two implications

$$R \Rightarrow P1$$

$$B \Rightarrow E(A) \quad .$$

In our particular case, it has been given, that this holds for $A = Q$ and $B = Q$, i.e. we know

$$Q \Rightarrow E(Q) \quad .$$

When we are now able to show that

$$fH_j(T) \Rightarrow E(fH_{j-1}(T)) \tag{20}$$

then $Q \text{ and } fH_j(T) \Rightarrow E(Q \text{ and } fH_{j-1}(T))$,

and as a result $Q \text{ and } fH_{j-1}(T) \Rightarrow fH'(R)$ is then a sufficient assumption about fH' to conclude that $Q \text{ and } fH_j(T) \Rightarrow fH"(R)$. As fH_j depends on fH_{j-1} as H" on H', this would conclude the induction step and (18) would have been proved.

We have two holes to fill: we have to show (20) and we have to extend the line of reasoning to texts of H2, containing more than one reference to H'. Let us first concentrate on (20).

We have defined $fH_j = G(fH_{j-1})$, but because for any P, we have $fH_{j-1}(P) \Rightarrow fH_{j-1}(T)$, an identical definition would have been

$$fH_j = G(\ fH_{j-1}(T) \text{ and } fH_{j-1})$$

i.e. each predicate formed by applying fH_{j-1} is replaced by its conjunction with $fH_{j-1}(T)$. And therefore, instead of

$P1 = fS(T)$ (i.e. P1 is the argument supplied to fH' in the evalu-
$P2 = fH_{j-1}(P1)$ ation of $fH''(T)$.)
$fH_j(T) = E(P2)$

we could have written equally well

$P1 = fS(T)$
$P2 = fH_{j-1}(P1)$
$fH_j(T) = E(P2 \text{ and } fH_{j-1}(T))$.

But $\{P2 \text{ and } fH_{j-1}(T)\} \Rightarrow fH_{j-1}(T)$ and therefore, because the transformer E enjoys the fifth property, we are entitled to conclude

$$fH_j(T) \Rightarrow E(fH_{j-1}(T)) \qquad \text{i.e. relation (20).}$$

To fill the second hole, viz. that in the text called H" more than one reference to Hʻ may occurr, is easier. Working backwards in the evaluation of $fH''(R)$ means that we first encounter the innermost evaluation(s) of fH', whose argument does not contain fH'. For those proposition transformers we apply our previous argument, showing that for them the weaker assumption $Q \text{ and } fH_{j-1}(T) \Rightarrow fH'(R)$ is sufficient. Then its value is replaced by Q and we start afresh. In this way the sufficiency of the weaker assuption about fH' can be established for all occurences of fHʻ -only a finite number!- in turn.

* * *

For the recursive routines of the particularly simple form

proc H: if B then S1; H else fi

we can ask ourselves what must be known about B and S1, when we take for R the special form Q and non B. Then

$$fH''(Q \text{ and non } B) = \{B \text{ and } fS1(fH'(Q \text{ and non } B))\} \text{ or } \{Q \text{ and non } B\}\ .$$

In order to be able to conclude $Q \Rightarrow fH''(Q \text{ and non } B)$ on account of $Q \Rightarrow fH'(Q \text{ and non } B)$, the necessary and sufficient assumption about fS1 is

$$\{Q \text{ and } B\} \Rightarrow fS1(Q). \qquad (21)$$

Procedures of this simple form are such useful elements that it is generally felt justified to introduce a specific notaion for it, in which the recursive procedure remains anonymous: it should contain as "parameters" the B and the S1 and we usually write

$$\text{while } B \text{ do } S1 \text{ od} \qquad (22)$$

When the statement S is of the form (22), we have now proved that

$$\{Q \text{ and } B\} \Rightarrow fS1(Q) \text{ implies } \{Q \text{ and } fS(T) \Rightarrow fS(Q \text{ and non } B).$$

This is called "The fundamental Invariance Theorem for Repetition".

* * *

The extension to a set of recursive procedures should not present any essential new difficulty. Given the texts

proc H1: .....H1.....H2.....H1.....H2.....corp

proc H2: .....H2.....H1.....H2.....corp,

the combined semantics are given by

$fH1_0 = fSTOP$ and $fH2_0 = fSTOP$, while for ~~XXX~~ $i > 0$:

$fH1_i = G1(fH1_{i-1}, fH2_{i-1})$ and $fH2_i = G2(fH1_{i-1}, fH2_{i-1})$. All our arguments can then be repeated, only twice as complicated. The Theorem of Invariance for the Recursive procedures is then as follows:

When there exist specific predicates Q1, R1, Q2 and R2, such that $Q1 \Rightarrow fH1'(R1)$ and $Q2 \Rightarrow fH2'(R2)$ are sufficient assumptions about fH1' and fH2' to prove $Q1 \Rightarrow fH1''(R1)$ and $Q2 \Rightarrow fH2''(R2)$, then we are allowed to conclude that $\{Q1 \text{ and } fH1(T)\} \Rightarrow fH1(R1)$ and $\{Q2 \text{ and } fH2(T)\} \Rightarrow fH2(R2)$.

This extension we gladly leave to the over-industrious reader. Among other things it implies that our results concerning bodies composed via concatenation and selection carries over to bodies also composed via repetition.

29 April 1973

End of EWD367