Finding the correctness proof of a concurrent program.

Introduction.

In this paper we want to do more than just giving another --be it unusual-- example of the utility of the first-order predicate calculus in proving the correctness of programs. In addition we want to show how, thanks to a systematic use of the first-order predicate calculus, fairly general --almost "syntactic"-- considerations about the formal manipulations involved can provide valuable guidance for the smooth discovery of an otherwise surprising argument.

For proofs of program correctness two fairly different styles have been developed, "operational" proofs and "assertional" proofs. Operational correctness proofs are based on a model of computation, and the corresponding computational histories are the subject matter of the considerations. In assertional correctness proofs the possibility of interpreting the program text as executable code is ignored and the program text itself is the subject matter of the formal considerations.

Operational proofs --although older and, depending on one's education, perhaps more "natural" than assertional proofs-- have proved to be tricky to design. For more complicated programs the required classification of the possible computational histories tends to lead to an exploding case analysis in which it becomes very clumsy to verify that no possible sequence of events has been overlooked, and it was in response to the disappointing experiences with operational proofs that the assertional style has been developed.

The design of an assertional proof --as we shall see below-- may present problems, but, on the whole, experience seems to indicate that assertional proofs are much more effective than operational ones in reducing the gnawing uncertainty whether nothing has been overlooked. This experience, already gained while dealing with sequential programs, was strongly confirmed while dealing with concurrent programs: the circumstance that the ratios of the speeds with which the sequential components proceed is left undefined greatly increases the class of computational histories that an operational argument would have to cover!

In the following we shall present the development of an assertional correctness proof of a program of N-fold concurrency. The program has been taken from the middle of a whole sequence of concurrent programs of increasing complexity --the greater complexity at the one end being the consequence of finer grains of interleaving-- . For brevity's sake we have selected here from this sequence the simplest item for which the assertional correctness proof displays the characteristic we wanted to show. (It is not the purpose of this paper to provide supporting material in favour of the assertional style: in fact, our example is so simple that an operational proof for it is still perfectly feasible.)

* * *

In the following y denotes a vector of N components $y[i]$ for $0 \leq i < N$. With the identifier f we shall denote a vector-valued function of a vector-valued argument, and the algorithm concerned solves the equation

$$y = f(y) \tag{1}$$

or, introducing $f_0, f_1, f_2, \ldots$ for the components of f

$$y[i] = f_i(y) \quad \text{for} \quad 0 \leq i < N \quad . \tag{2}$$

It is assumed that the initial value of y and the function f are such that repeated assignments of the form

$$< y[i] := f_i(y) > \tag{3}$$

will lead in a finite number of steps to y being a solution of (1). In (3) we have used Lamport's notation of the angle brackets: they enclose "atomic actions" which can be implemented by ensuring between their executions mutual exclusion in time. For the sake of termination we assume that the sequence of i-values for which the assignments (3) are carried out is (the proper begin of) a sequence in which each i-value occurs infinitely often. (We deem this property guaranteed by the usual assumption of "finite speed ratios"; he who refuses to make that assumption can read the following as a proof of partial correctness.)

For the purpose of this paper it suffices to know that functions f exist such that with a proper initial value of y equation (1) will be solved

by a finite number of assignments (3). How for a given function f and initial value y this property can be established is not the subject of this paper. (He who refuses to assume that the function f and the initial value of y have this property is free to do so: he can, again, read the following as a proof of partial correctness that states that when our concurrent program has terminated, (1) is satisfied.)

Besides the vector y there is --for the purpose of controlling termination-- a vector h, with boolean elements h[i] for $0 \leq i < N$, all of which are true to start with. We now consider the following program of N-fold concurrency, in which each atomic action assigns a value to at most one of the array elements mentioned. We give the program first and shall explain the notation afterwards.

The concurrent program we are considering consists of the following N components $cpnt_i$ $(0 \leq i < N)$:

$cpnt_i$:

```
    L0:  do < (E j: h[j]) > →
    L1:          < if y[i] = f_i(y) → h[i]:= false >
                    [] y[i] ≠ f_i(y) → y[i]:= f_i(y) > ;
    L2j:                                (A j: < h[j]:= true > )
                 fi
         od
```

In line L0, "(E j: h[j])" is an abbreviation for

$$(\underline{E}\ j:\ 0 \leq j < N:\ h[j]) \quad ;$$

for the sake of brevity we shall use this abbreviation throughout this paper. By writing $< (\underline{E}\ j:\ h[j]) >$ in the guard we have indicated that the inspection whether a true h[j] can be found is an atomic action.

The opening angle bracket " < " in L1 has two corresponding closing brackets, corresponding to the two "atomic alternatives"; it means that in the same atomic actions the guards are evaluated and either "h[i]:= false" or "$y[i]:= f_i(y)$" is executed. In the latter case, N separate atomic actions follow, each setting an h[j] to true: in line L2j we have used

the abbreviation "(A j: < h[j]:= true >)" for the program that performs the N atomic actions < h[0]:= true > through < h[N-1]:= true > in some order which we don't specify any further.

In our target state y is a solution of (1), or, more explicitly

$$(\underline{A}\ j:\ y[j] = f_j(y)) \qquad (4)$$

holds. We first observe that (4) is an invariant of the repeatable statements, i.e. once true it remains true. In the alternative constructs always the first atomic alternative will then be selected, and this leaves y , and hence (4) unaffected. We can even conclude a stronger invariant

$$\underline{non}\ (\underline{E}\ j:\ h[j])\ \underline{and}\ (\underline{A}\ j:\ y[j] = f_j(y)) \qquad (5)$$

or, equivalently

$$(\underline{A}\ j:\ \underline{non}\ h[j])\ \underline{and}\ (\underline{A}\ j:\ y[j] = f_j(y)) \qquad (5')$$

for, when (5) holds, no assignment h[i]:= false can destroy the truth of (A j: non h[j]) . When (4) holds, the assumption of finite speed ratios implies that within a finite number of steps (5) will hold. But then the guards of the repetitive constructs are false, and all components will terminate nicely with (4) holding. The critical point is: can we guarantee that none of the components terminates too soon?

We shall give an assertional proof, following the technique which has been pioneered by Gries and Owicki [1]. We call an assertion "universally true" if and only if it holds between any two atomic actions --i.e. "always" with respect to the computation, "everywhere" with respect to the text-- . More precisely: proving the universal truth of an assertion amounts to showing

1) that it holds at initialization
2) that its truth is an invariant of each atomic action.

In order to prove that none of the components terminates too soon, i.e. that termination implies that (4) holds, we have to prove the universal truth of

$$(\underline{E}\ j:\ h[j])\ \underline{or}\ (\underline{A}\ j:\ y[j] = f_j(y)) \qquad . \qquad (6)$$

Relation (6) certainly holds when the N components are started because initially all h[j] are true. We are only left with the obligation to prove the invariance of (6); the remaining part of this paper is devoted to that proof, and to how it can be discovered.

We get a hint of the difficulties we may expect when trying to prove the invariance of (6) with respect to the first atomic alternative of L1:

$$< y[i] = f_i(y) \rightarrow h[i] := \text{false} >$$

as soon as we realize that the first term of (6) is a compact notation for

$$h[0] \text{ or } h[1] \text{ or } \ldots \text{ or } h[N-1]$$

which only changes from true to false when, as a result of "h[i]:= false" the last true h[j] disappears. That is ugly!

We often prove mathematical theorems by proving a stronger --but, somehow, more manageable-- theorem instead. In direct analogy: instead of trying to prove the invariant truth of (6) directly, we shall try to prove the invariant truth of a stronger assertion that we get by replacing the conditions $y[j] = f_j(y)$ by stronger ones. Because non R is stronger than Q provided (Q or R) holds, we can strengthen (6) into

$$(\underline{E}\ j:\ h[j]) \text{ or } (\underline{A}\ j:\ \text{non } R_j) \qquad (7)$$

provided

$$(\underline{A}\ j:\ y[j] = f_j(y) \text{ or } R_j) \qquad (8)$$

holds. (Someone who sees these heuristics presented in this manner for the first time may experience this as juggling, but I am afraid that it is quite standard and that we had better get used to it.)

What have we gained by the introduction of the N predicates R_j ? Well, the freedom to choose them! More precisely: the freedom to define them in such a way that we can prove the universal truth of (8) --which is structurally quite pleasant-- in the usual fashion, while the universal truth of (7) --which is structurally equally "ugly" as (6)-- follows more or less directly from the definition of the R_j's : that is the way in which we may hope that (7) is more "manageable" than the original (6).

In order to find a proper definition of the R_j's, we analyse our obligation to prove the invariance of (8).

If we only looked at the invariance of (8), we might think that a definition of the R_j's in terms of y :

$$R_j = (y[j] \neq f_j(y))$$

would be a sensible choice. A moment's reflection tells us that that definition does not help: it would make (8) universally true by definition, and the right-hand terms of (6) and (7) would be identical, whereas under the truth of (8), (7) was intended to be stronger than (6).

For two reasons we are looking for a definition of the R_j's in which the y does not occur: firstly, it is then that we can expect the proof of the universal truth of (8) to amount to something --and, thereby, to contribute to the argument-- , secondly, we would like to conclude the universal truth of (7) --which does not mention y at all!-- from the definition of the R_j's . In other words, we propose a definition of the R_j's which does not refer to y at all: only with such a definition does the replacement of (6) by (7) and (8) localize our dealing with y completely to the proof of the universal truth of (8).

Because we want to define the R_j's independently of y , because initially we cannot assume that for some j-value $y[j] = f_j(y)$ holds, and because (8) must hold initially, we must guarantee that initially

$$(\underline{A}\ j:\ R_j) \qquad (9)$$

holds. Because, initially, all the $h[j]$ are true, the initial truth of (9) is guaranteed if the R_j's are defined in such a way that we have

$$(\underline{E}\ j:\ \underline{non}\ h[j])\ \underline{or}\ (\underline{A}\ j:\ R_j) \quad . \qquad (10)$$

We observe, that (10) is again of the recognized ugly form we are trying to get rid of. We have some slack --that is what the R_j's are being introduced for-- and this is the moment to decide to try to come away with a stronger --but what we have called: "structurally more pleasant"-- relation for the definition of the R_j's , from which (10) immediately follows. The only candidate I can think of is

$$(\underline{A}\ j:\ \underline{non}\ h[j]\ \underline{or}\ R_j) \qquad (11)$$

and we can already divulge that, indeed, (11) will be one of the defining equations for the R_j's .

From (11) it follows that the algorithm will now start with all the

R_j's true. From (8) it follows that the truth of R_j can be appreciated as "the equation $y[j] = f_j(y)$ need not be satisfied", and from (7) it follows that in our final state we must have all the R_j's equal to false.

Let us now look at the alternative construct

L1: $<$ **if** $y[i] = f_i(y) \rightarrow h[i] := \text{false} >$
$\quad [] \; y[i] \neq f_i(y) \rightarrow y[i] := f_i(y) > ;$
L2j: $\quad (\underline{A}\ j: < h[j] := \text{true} >)$
fi .

We observe that the first alternative sets $h[i]$ false, and that the second one, as a whole, sets all $h[j]$ true. As far as the universal truth of (11) is concerned, we therefore conclude that in the first alternative R_i is allowed to, and hence <u>may</u> become false, but that in the second alternative as a whole, all R_j's <u>must</u> become true.

Let us now confront the two atomic alternatives with (8). Because, when the first atomic alternative is selected, only $y[i] = f_i(y)$ has been observed, the universal truth of (8) is guaranteed to be an invariant of the first atomic alternative, provided it enjoys the following property (12):

In the execution of the first atomic alternative

$$< y[i] = f_i(y) \rightarrow h[i] := \text{false} >$$

<u>no</u> R_j for $j \neq i$ changes from true to false. (12)

Confronting the second atomic alternative

$$< y[i] \neq f_i(y) \rightarrow y[i] := f_i(y) >$$

with (8), and observing that upon its completion <u>none</u> of the relations $y[j] = f_j(y)$ needs to Hold, we conclude that the second atomic alternative itself must already cause a final state in which all the R_j's are true, in spite of the fact that the subsequent assignments $h[j] := \text{true}$ --which would each force an R_j to true on account of (11)-- have not been executed yet. In short: in our definition for the R_j's we must include besides (11) <u>another</u> reason why an R_j should be defined to be true.

As it stands, the second atomic alternative only modifies y, but we had decided that the definition of the R_j's would not be expressed in terms

of y ! The only way in which we can formulate the additional reason for an R_j to be true is in terms of an auxiliary variable (to be introduced in a moment), whose value is changed in conjunction with the assignment to $y[i]$. The value of that auxiliary variable has to force each R_j to true until the subsequent assignment $< h[j] := \text{true} >$ does so via (11). Because the second atomic alternative is followed by N subsequent, separate atomic actions $< h[j] := \text{true} >$ --one for each value of j -- , it stands to reason that we introduce for the i-th component $cpnt_i$ an auxiliary local boolean array s_i with elements $s_i[j]$ for $0 \leq j < N$. Their initial (and "neutral") value is true. The second atomic alternative of L1 sets them all to false, the atomic statements L2j will reset them to true one at a time.

In contrast to the variables y and h, which are accessible to all components --which is expressed by calling them "global variables"-- , each variable s_i is only accessible to its corresponding component $cpnt_i$ --which is expressed by calling the variable s_i "local" to component $cpnt_i$-- .

Local variables give rise to so-called "local assertions". Local assertions are most conveniently written in the program text of the individual components at the place corresponding to their truth: they state a truth between preceding and succeeding statements in exactly the same way as is usual in annotating or verifying sequential programs. If a local assertion contains only local variables, it can be justified on account of the text of the corresponding component only.

In the following annotated version of $cpnt_i$ we have inserted local assertions between braces. In order to understand the local assertions about s_i it suffices to remember that s_i is local to $cpnt_i$. The local assertion $\{R_i\}$ in the second atomic alternative of L1 is justified by the guard $y[i] \neq f_i(y)$, in conjunction with (8). We have further incorporated in our annotation the consequence of (12) and the fact that the execution of a second alternative will never cause an R_j to become false: a true R_i can only become false by virtue of the execution of the first alternative of L1 by $cpnt_i$ itself! Hence, R_i is true all through the execution of the second alternative of $cpnt_i$.

```
cpnt_i:
    L0:  do < ( E j: h[j] > → {(A j: s_i[j])}
    L1:       < if y[i] = f_i(y) → h[i]:= false > {A j: s_i[j]}
                 [] y[i] ≠ f_i(y) →
                      {R_i} y[i]:= f_i(y);
                      (A j: s_i[j]:= false) > {R_i and (A j: non s_i[j])};
    L2j:              (A j:{R_i and non s_i[j]} < h[j]:= true; s_i[j]:= true > )
                 fi {(A j: s_i[j])}
    od
```

On account of (11) R_j will be true upon completion of L2j . But the second atomic alternative of L1 should already have made R_j true, and it should remain so until L2j is executed. The precondition of L2j, as given in the annotation, hence tells us the "other reason besides

$$(\underline{A}\ j:\ \underline{non}\ h[j]\ \underline{or}\ R_j) \qquad (11)$$

why an R_j should be defined to be true":

$$(\underline{A}\ i,\ j:\ \underline{non}\ R_i\ \underline{or}\ s_i[j]\ \underline{or}\ R_j) \qquad (13)$$

Because it is our aim to get eventually all the R_j's false, we <u>define</u> the R_j's as the <u>minimal</u> solution of (11) and (13), minimal in the sense of: as few R_j's true as possible.

The existence of a unique minimal solution of (11) and (13) follows from the following construction. Start with all R_j's false --all equations of (13) are then satisfied on account of the term "<u>non</u> R_i" -- . If all equations of (11) are satisfied as well, we are ready --no true R_j's at all-- ; otherwise (11) is satisfied by setting R_j to true for all j-values for which $h[j]$ holds. Now all equations of (11) are satisfied, but some of the equations of (13) need no longer be satisfied: as long as an (i, j)-pair can be found for which the equation of (13) is not satisfied, satisfy it by setting that R_j to true: as this cannot cause violation of (11) we end up with the R_j's being a solution of (11) and (13). But it is also the minimal solution, because any R_j true in this solution must be true in any solution.

For a value of i , for which

$$(\underline{A}\ j:\ s_i[j]) \qquad (14)$$

holds, the above construction tells us that the truth of R_i forces no further true R_j's via (13); consequently, when such an R_i becomes false, no other R_j-values are then affected. This, and the fact that the first atomic alternative of L1 is executed under the truth of (14) tells us, that with our definition of the R_j's as the minimal solution of (11) and (13), requirement (12) is, indeed, met.

We have proved the universal truth of (8) by defining the R_j's as the minimal solution of (11) and (13). The universal truth of (7), however, is now obvious. If the left-hand term of (7) is false, we have

$$(\underline{A}\ j:\ \underline{non}\ h[j]),$$

and (11) and (13) have as minimal solution all R_j's false, i.e.

$$(\underline{A}\ j:\ \underline{non}\ R_j)$$

which is the second term of (7). From the universal truth of (7) and (8), the universal truth of (6) follows, and our proof is completed.

Concluding remarks.

This note has been written with many purposes in mind:

1) To give a wider publicity to an unusual problem and the mathematics involved in its solution.

2) To present a counterexample contradicting the much-propagated and hence commonly held belief that correctness proofs for programs are only laboriously belabouring the obvious.

3) To present a counterexample to the much-propagated and hence commonly held belief that there is an antagonism between rigour and formality on the one hand and "understandability" on the other.

4) To present an example of a correctness proof in which the first-order predicate calculus is used as what seems an indispensable tool.

5) To present an example of a correctness proof in which the first-order predicate calculus is a fully adequate tool.

6) To show how fairly general --almost "syntactic"-- considerations about the formal manipulations involved can provide valuable guidance for the discovery of a surprising and surprisingly effective argument, thus showing how a formal discipline can assist "creativity" instead of --as is sometimes suggested-- hampering it.

7) To show how also in such formal considerations the principle of separation of concerns can be recognized as a very helpful one.

I leave it to my readers to form their opinion whether with the above I have served these purposes well.

Acknowledgements. I would like to express my gratitude to both IFIP WG2.3 and "The Tuesday Afternoon Club", where I had the opportunity to discuss this problem. Those familiar with the long history that led to this note, however, know that in this case I am indebted to C.S.Scholten more than to anyone else. Comments from S.T.M.Ackermans, David Gries, and W.M.Turski on an earlier version of this paper are greatfully acknowledged.

[1] Owicki, Susan and Gries, David, "Verifying Properties of Parallel Programs: An Axiomatic Approach". Comm.ACM 19, 5 (May 1976), pp.279-285.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow