

ASPECTS OF REASONING EFFECTIVELY ABOUT DISTRIBUTED SYSTEMS

E.W. Dijkstra

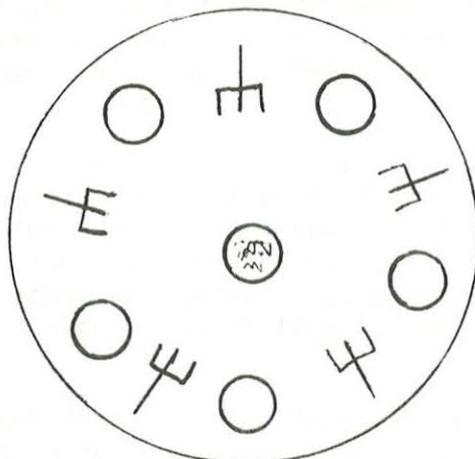
Rapporteurs : Mr. E. Best
Mr. P. Smith

The consideration of "distributed systems" certainly adds a new dimension to the types of complications we must be able to cope with, and I intend to show two examples of arguments that have proved effective in reasoning about distributed systems. The first argument is relatively simple; it is derived from a problem statement which might strike you as very artificial. It is however, thanks to its generality, a fairly powerful model of all sorts of exclusion constraints. The argument as presented here is the result of at least five iterations, and by now it is a pleasure to present because it is very beautiful.

I made this remark because I wanted to stress the need for our arguments to be both understood and understandable; this I consider not a dispensible luxury, but a necessity if we are to avoid making a mess out of computer science.

The Problem of the Dining Philosophers Revisited

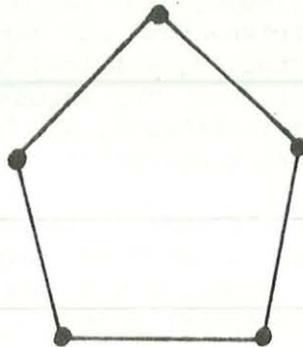
The example I am going to present is a generalisation of the by now traditional problem of the five dining philosophers. We imagine a round table with five plates, five forks between the plates, and five philosophers in front of the plates:



The life of a philosopher consists of an alternation of thinking and eating, these being two mutually exclusive states. The dish served is a difficult kind of spaghetti for which two forks are needed; now there are two forks next to each plate, so there is no problem. However, if one envisages an asymmetrical solution in which each philosopher grabs his left hand fork first whenever he wishes to eat, and then turns his attention to his right hand fork, grabs it, starts eating and puts down his two forks, then of course a deadlock is possible if all five get hungry simultaneously. So then one devises another solution in which a philosopher can grab both his forks simultaneously, provided he is hungry and both forks are free. This solution maximises resource utilisation: forks are

only grabbed when really used for eating. However, this is not too attractive a solution, either, because a philosopher could be starved to death by a conspiracy of his neighbours. We may conclude - and this is a good thing to know - that the two goals of high resource utilisation and quick response may be quite incompatible.

The problem is quite simply generalised. With the above formulation, we may represent each mutual constraint, that is, the forks, as an edge of a graph the nodes of which correspond to the philosophers. Obviously, we then get a pentagon:



Philosophers sitting at nodes which are connected by an edge may be called each other's neighbours, and the rule is: no two neighbours eat simultaneously.

By just adding or deleting edges, we are able to model all sorts of mutual exclusion constraints. For example, if we augment the pentagon to the complete graph of five nodes, we have the situation where each philosopher excludes each other philosopher, that is, at most one of five philosophers eats at a time.

In general, we may now consider an arbitrary but finite graph, with a philosopher of the same kind as before sitting at each node. Directly connected philosophers are called each other's neighbours, and neighbours are not allowed to eat simultaneously. We are requested to solve this synchronisation problem in such a way that neither deadlock nor starvation in the above mentioned sense will arise.

One property of the solution is evident from the outset: since the problem formulation is independent of the shape of the graph, so should be the argument leading up to the solution. Consequently, it should be possible to study an arbitrary philosopher's behaviour relating to his neighbours to such an extent that we gain the basis for an induction argument over the graph - whatever its shape may be.

Furthermore, in order to come to grips with the requirement that no two neighbours eat simultaneously, I am going to first express "simultaneity between neighbours"; I will do this by giving each philosopher two states, "thinking" and "tabled". For the time being there will be no synchronisation constraints; there will be a marking of the edges, indicating whether or not the two

neighbours are simultaneously tabled. In short, I wish to maintain the universal truth of

\forall neighbours A and B:

"A and B both tabled" \equiv "edge A-B marked" (P1)

Initially, all philosophers are thinking, and (P1) can be made true by starting off with unmarked edges.

Now it remains to be ensured that whenever the state of a philosopher changes, the appropriate markers of edges are introduced or removed, respectively. We have to consider two transitions, T1 and T2; T1 changes a philosopher's state from "thinking" to "tabled" while T2 changes his state from "tabled" to "thinking".

Let me consider T1 first; in changing from "thinking" to "tabled", a philosopher may introduce pairs of tabled neighbours. Hence T1 must effect the marking of all edges which lead to his tabled neighbours, in order that the invariance of (P1) is maintained. In summary, T1 becomes:

<mark edges connecting you to your tabled neighbours, and switch from "thinking" to "tabled">.

The angle brackets around this action mean that we are to regard T1 as an "atomic action" (for which see the manuscript of the next problem).

Now let me consider T2; quite simply, all markers on edges leading to tabled neighbours have to be removed to ensure (P1). Therefore, T2 is

<remove all markers from edges connecting you to tabled neighbours, and switch from "tabled" to "thinking">.

As a consequence of these definitions, (P1) is invariantly true.

Next, we observe that a marked edge has always been marked by one of the two neighbours, namely by the one who was the last of the two to carry out T1. As it stands, our symbolism does not express this; my next step will remedy the situation by introducing two forms of marking of an edge: an arrow in one or the other direction. This can be used for coding which of the two neighbours places the marker. We decide that a marker should be placed as an outgoing arrow; hence T1 now becomes:

<direct arrows towards all your tabled neighbours, and switch from "thinking" to "tabled">.

On the other hand, T2 becomes

<remove all arrows along edges connecting you to tabled neighbours, and switch from "tabled" to "thinking">.

I will now split the state "tabled" into two successive states: first "hungry" then "eating". Between the two, there will be a transition T1.5, saying

<when without outgoing arrows, switch from "hungry" to "eating">.

Observe that each philosopher is the sole creator of his outgoing arrows; all his neighbours can inflict on him are arrows pointing towards him. Now since outgoing arrows are introduced solely by T1, and since T1.5 only takes place when there are no outgoing arrows, it follows that a philosopher is without outgoing arrows while "eating" or "thinking". That is, we have just proved that

\forall philosophers A:

(P2)

A is hungry or A is without outgoing arrows

is invariantly true (provided it is true to start with, which it is since there are no arrows at all).

Now consider (P1) and (P2) in conjunction. We consider a pair A,B of tabled neighbours. From (P1) it follows that there is a marker, that is, an arrow, between A and B; from (P2) it follows that either A or B (or both) are hungry. In other words, the requirement that no two neighbours are eating simultaneously is satisfied.

There remains to be proved the absence of deadlock and individual starvation. I am required to prove that each hungry philosopher will eventually start eating, that is, execute action T1.5. I do so by first showing that the following is invariantly true:

The arrows never form a directed cycle

(P3)

Initially, (P3) is true since there are no arrows. Arrows may be introduced by T1; but according to (P1), a "thinking" philosopher has only unmarked edges. Therefore, T1 can never close a directed cycle. Since T1 is the only means of arrow creation, it follows that (P3) remains true.

I will use (P3) to show that indeed each hungry philosopher will eventually have his turn in performing T1.5. What is more, an upper bound for his period of delay in the state "hungry" can be given. I will make use of three assumptions. Firstly, we recall our assumption that the graph is finite; secondly, we assume that once a philosopher is without outgoing arrows, he will perform T1.5 immediately; thirdly, we assume that each eating session of a philosopher will be finite.

Consider a hungry philosopher. Paint his outgoing arrows red; collect the nodes reached by the red arrows and paint their outgoing arrows red; repeat until no longer applicable. Termination is guaranteed because of (P3) and the finiteness of the graph. The outcome will be a directed subtree of the graph, painted red, whose root consists of the original hungry philosopher.

Obviously, the philosophers at the leaves of the red tree do not have outgoing arrows and therefore are already eating. Nor can they introduce any outgoing arrows while they are eating. On

account of this, the red tree does not grow. Instead, it will shrink to a smaller tree when, after some finite time, the leaf philosophers stop eating and remove their incoming arrows, including the red ones. By induction, after a finite time all outgoing arrows of the original philosopher will be erased, which means that he will then start eating.

This completes the proof that deadlock and individual starvation are absent in the solution. From the proof we can derive an upper time limit for a philosopher to be delayed in state "hungry":

K^* "maximum eating time of his colleagues",

where K is the maximum reachability path in the graph.

This ends the argument. Why is it so effective? I think the argument is beautiful for a number of reasons:

Firstly, we did not need to take into account certain intractable aspects of the problem statement such as the shape of the graph.

Furthermore - and this point will be stressed even more in my next example - I found it very convenient to formulate invariants as symmetrical expressions of their arguments. For example, (P2) is an or - clause, not an implication. On the one hand, one can easily derive from it the implications both ways ($A \vee B \equiv \neg A \Rightarrow B \equiv \neg B \Rightarrow A$); on the other hand, the invariance proof of (P2) becomes much more uniform once (P2) is stated in its symmetrical form.

Thirdly, we could use "stepwise refinement". At the beginning the collective state "tabled" and the unrefined symbol "marked" already sufficed to prove (P1), and we were able to postpone the introduction of complications to the moment when they really had to be considered.

Discussion

Dr. Whitby-Strevens : You started off by saying that distributed computing is introducing an order of magnitude of difficulty. Are you not defining yourself away from some of this in your assumption that you deal with "atomic actions"?

Professor Dijkstra : You are right in the sense that when you consider that the actual constraints for ensuring the atomicity of the actions given do take time, then you will face exactly the mutual exclusion problem we started off to solve: no two neighbours are allowed to perform T1 simultaneously. So in a sense I have reduced the problem to exactly the same problem.

However, there is good reason for doing so. You see, there is another solution to the original problem, which I would describe as cheap and ugly: just number all edges from 1 to N , say, and introduce the single rule that each philosopher has to claim his edges in the order of increasing numbers. Then, again, the system is deadlock-free and starvation-free as before; however, the overall

behaviour strongly depends on the arbitrary numbering of edges. Thus I propose not to care about the implementation of the atomic action concept; we might imagine that this is done with a cheap but ugly technique. On the next level, however, where the mutual exclusion constraints are our main concern, we abstract away from the arbitrariness of such implementations and can therefore contemplate the more elegant and uniform solution just given which makes the overall system performance more overseable and predictable.

Finding the correctness proof of a concurrent program

Introducing his second example, Professor Dijkstra cautioned his audience that it might strain their abilities in the first order predicate calculus, but claimed that it was essential that they be able to teach this method to students. His presentation followed closely the paper which is reprinted in the sequel.

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

$$(1) \quad y = f(y)$$

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

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

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

$$(3) \quad \langle y[i] := f_i(y) \rangle$$

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 $cpnti$ ($0 \leq i < N$):

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

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

$$(Ej:0 \leq j < N:h[j]) \quad ;$$

for the sake of brevity we shall use this abbreviation throughout this paper. By writing $\langle(Ej:h[j])\rangle$ 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 " \langle " in L1 has two corresponding closing brackets, corresponding to the two "atomic alternatives"; it means that in the same atomic action the guards are evaluated and either " $h[i]:=false$ " or " $y[i]:=fi(y)$ " 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 " $(Aj:\langle h[j]:=true \rangle)$ " for the program that performs the N atomic actions $\langle h[0]:=true \rangle$ through $\langle h[N-1]:=true \rangle$ in some order which we don't specify any further.

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

$$(4) \quad (Aj:y[j]=fj(y))$$

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

$$(5) \quad \text{non } (Ej:h[j]) \text{ and } (Aj:y[j]=fj(y))$$

or equivalently

$$(5') \quad (Aj:\text{non } h[j]) \text{ and } (Aj:y[j]=fj(y))$$

for, when (5) holds, no assignment $h[i]:=false$ can destroy the truth of $(Aj:\text{non } h[j])$. 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 initialisation
- 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

$$(6) \quad (Ej:h[j]) \text{ or } (Aj:y[j]=fj(y)).$$

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 $L1$:

$$\langle y[i]=fi(y) \rightarrow h[i]:=false \rangle$$

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

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

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 $y[j]=fj(y)$ by stronger ones. Because $\text{non } R$ is stronger than Q provided $(Q \text{ or } R)$ holds, we can strengthen (6) into

$$(7) \quad (Ej:h[j]) \text{ or } (Aj:\text{non } Rj)$$

provided

$$(8) \quad (Aj:y[j]=fj(y) \text{ or } Rj)$$

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 Rj ? 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 as 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 :

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

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) localise 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

$$(9) \quad (A_j : R_j)$$

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

$$(10) \quad (E_j : \text{non } h[j]) \text{ or } (A_j : R_j).$$

We observe, that (10) is again of the recognised 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

$$(11) \quad (A_j : \text{non } h[j] \text{ or } R_j)$$

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]=fi(y) → h[i]:=false>
    [] y[i] fi(y) → y[i]:=fi(y)>;
L2j: (Aj:<h[j]:=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 may become false, but that in the second alternative as a whole, all R_j 's must become true.

Let us now confront the two atomic alternatives with (8). Because, when the first atomic alternative is selected, only $y[i]=fi(y)$ 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]=fi(y) → h[i]:=false>
```

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

Confronting the second atomic alternative

```
<y[i] fi(y) → y[i]:=fi(y)>
```

with (8), and observing that upon its completion none 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 $h[j]:=true$ - 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) another 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 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 $h[j]:=true$ does so via (11). Because the second atomic alternative is followed by N subsequent, separate atomic actions $h[j]:=true$ - one for each value of j -, it stands to reason that we introduce for the i -th component c_{pti} 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 $cpnti$ - which is expressed by calling the variable si "local" to component $cpnti$.

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 $cpnti$ we have inserted local assertions between braces. In order to understand the local assertions about si it suffices to remember that si is local to $cpnti$. The local assertion $\{R_i\}$ in the second atomic alternative of L_1 is justified by the guard $y[i] \neq fi(y)$ 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 L_1 by $cpnti$ itself! Hence, R_i is true all through the execution of the second alternative of $cpnti$.

$cpnti$:

```

L0: do <(Ej:h[j])> → {(Aj:si[j])}
L1:   <if y[i]=fi(y) → h[i]:=false>{Aj:si[j]}
      [] y[i]≠fi(y) →
        {Ri}y[i]:=fi(y);
        (Aj:si[j]:=false)>{Ri and (Aj:non si[j])};
L2j:   (Aj:{Ri and non si[j]}<h[j]:=true;
        si[j]:=true>)
      fi {(Aj:si[j])}
      od

```

On account of (11) R_j will be true upon completion of L_{2j} . But the second atomic alternative of L_1 should already have made R_j true, and it should remain so until L_{2j} is executed. The precondition of L_{2j} , as given in the annotation, hence tells us the "other reason besides

$$(11) \quad (Aj:\text{non } h[j] \text{ or } R_j)$$

why an R_j should be defined to be true":

$$(13) \quad (A_{i,j}:\text{non } R_i \text{ or } si[j] \text{ or } R_j).$$

Because it is our aim eventually to get all the R_j 's false, we define the R_j 's as the minimal 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 "non 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

$$(14) \quad (A_j:si[j])$$

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 L_1 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

$$(A_j:\text{non } h[j]),$$

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

$$(A_j:\text{non } R_j)$$

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 recognised 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 gratefully acknowledged.

Reference

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

Discussion

Professor Pyle asked whether it was necessary to prove the existence of the Rj's or whether it was obvious. Professor Dijkstra replied that the proof of existence had been done in the paper he was presenting. He then went on to make a more general point concerning the discovery of an analytical calculus for geometry by Rene Descartes. Prior to the existence of analytical proofs, an 'elusive form of invention' was required to find auxiliary construction lines. Descartes made an error of judgement in thinking that his method eliminated intuition. We have since found that in order to keep proofs within manageable proportions it is often necessary to invent auxiliary variables. Although his method offers strong heuristics for this, the power of invention is still required. Professor Pyle pointed out that one of the causes of fallacious geometric arguments was the use of non-existent construction lines.

Mr. Shelness referred to an incident at the University of Edinburgh whereby examination students were given a program and asked to generate a proof of correctness; finally they were asked why their proof was better than testing the program. The question was answered by all but one member of the class who pointed out that

the program was incorrect. The reasoning which generated the incorrect program also generated the incorrect proof. Mr. Shelness said that he thought we should ask ourselves whether understanding the proof was simpler than understanding the program. Professor Dijkstra replied that he had done his best to reduce the length of the proof of his second example and that it now required only five lines. The proof had not played a role in the process of programming although this was his 'beloved way of program construction'.

The following table shows the results of the survey conducted in the year 1998. The data is presented in a tabular format, with the first column representing the different categories of respondents, and the second column representing the percentage of respondents in each category. The total number of respondents is 1000.

Category	Percentage
Male	55%
Female	45%
Age Group 18-25	15%
Age Group 26-35	25%
Age Group 36-45	20%
Age Group 46-55	20%
Age Group 56-65	10%
Age Group 66+	5%
Education Level: High School	30%
Education Level: Bachelor's Degree	40%
Education Level: Master's Degree	15%
Education Level: Doctorate	10%
Other	5%