Some correctness proofs for the Safe Sluice

This note is written for my own file mainly. It is a stepping stone for a reinvestigation of the usefulness of the Gries - Owicki theory for demonstrating the partial correctness of concurrent programs. The reinvestigation starts with an exploration of very small multiprograms, of which the Safe Sluice is an example.

There are quite a few indications for the Gries - Owicki theory to be repudiated, such as G.L. Peterson disliking a correctness proof for his famous algorithm, such as M. Raynal publishing a book on mutual exclusion algorithms without any reference to Gries - Owicki, and such as the proliferation of Temporal Logic. The reason for the repudiation is presumably threefold. First, the theory can not cope with total correctness. Second, the theory can only be used (= had only been used) to provide an aposteriori argument. Third, the sizes of the proofs grow exponentially with the sizes of the programs.

Over the last few years, however, our mastery of the predicate calculus has increased by at least one order of magnitude. Also, computing scientists have become more keen on consciously separating their concerns and they have become more alert on not getting entangled in twisted mathematical contraptions. Armed with these acquirements and buttressed by a number of encouraging examples from the recent past, we reinvestigate the usefulness of the Gries - Owicki theory.

*

The Gries- Owicki theory is one of the earliest theories for discussing the partial correctness of multi-programs. It can be summarized as follows. Consider a system of annotated sequential programs. The annotation is such that it provides a precondition for each of the constituent atomic statements. For the annotation to be correct, we have to show that

i) each program's initial assertion is satisfied by the initial state of the system
ii) it is correct for each program in isolation (local correctness); this is done by showing that each pattern $\{P\}\ S\ \{Q\}$ within a program satisfies $P \Rightarrow wlp.(S, Q)$.
iii) it is correct for the programs in cooperation (global correctness); this is done by showing that each assertion P taken from one program and each pattern $\{Q\}\ S$ taken from a distinct program satisfies $P \wedge Q \Rightarrow wlp.(S, F$

The mechanical interpretation of a correctly annotated text is that whenever a statement within a program is selected for execution, the system's state satisfies the statement's precondition. The only virtue of the mechanical interpretation is that it enables us to "model intended behaviours".

+

The Safe Sluice is the name which has been attached to one of the earliest solutions to the problem of realizing mutual exclusion within a

critical section for a system of two programs. The solution can be described as follows. The two programs p and q operate on the two booleans x.p and x.q, which are false initially. Program p is a cyclic program with body

```
prog.p :   x.p := true
       ; [¬x.q → skip ]
       ; CS.p
       ; x.p := false .
```

Program q is program p with p and q interchanged

Remarks on notation

- Unless stated otherwise, each line of code is considered atomic. This means that in applying the Gries-Owicki theory, each line needs to be prefixed with an assertion.
- A statement $[B \to S]$ is short for do $\neg B \to$ skip od ; S. It is characterized by
  $wlp.([B \to S], R) \equiv \neg B \vee wlp.(S, R)$
- All programs we consider will be cyclic. Therefore we adopted the convention of just mentioning the body of the repetition. This might be very unwise because the convention does not release us -- the theorem of invariance being what it is -- from the obligation to show that the body's postcondition implies its precondition. We will retract the convention at the first occasion it causes difficulties of whatever kind.

(End of Remarks on notation.)

+

Next we use the Gries - Owicki theory to describe a number of correctness proofs for the Safe Sluice.

Proof 0

This is a very old proof. It emerged shortly after the Gries - Owicki theory. It is taken from EWD 554 ("A Personal Summary of the Gries - Owicki Theory"), not verbatim but as far as its shape is concerned. Its main characteristic is that it is constructed from an operational understanding of the algorithm. In that respect it might be typical of the way in which the Gries - Owicki theory has always been used.

The proof starts with the introduction of two boolean thought variables $y.p$ and $y.q$, which are initialized with the value false. The programs are modified with operations on them as follows.

```
prog.p :   {P0}   x.p := true
         ; {P1}   [¬ x.q → y.p := true]
         ; {P2}   CS.p
         ; {P2}   x.p, y.p := false, false
           {P0}
```

(prog.p is prog.q with p and q interchanged. We will not repeat this anymore.)

For the P_i's we can choose

$P0: \quad \neg x.p \wedge \neg y.p$

$P1: \quad x.p \wedge \neg y.p$

$P2: \quad x.p \wedge y.p$.

For these choices we can prove that the annotation is correct: it is obviously correct for program p in isolation, and because program q does not modify any of the variables mentioned in the P_i's, it is also correct for program p in cooperation.

Next we observe that each P_i implies P given b

$$P: \quad x.p \vee \neg y.p .$$

Therefore we can rely on the universal validity of P. By symmetry, we can also rely on the universal validity of

$$Q: \quad x.q \vee \neg y.q .$$

We are heading for the universal validity of

$$R: \quad \neg y.p \vee \neg y.q ,$$

which formalizes the intended behaviour, which is that no two programs are engaged in their critical sections simultaneously. The only statement that could falsify R is an assignment $y := true$. However,

$$\begin{aligned} & wlp.([\neg x.q \rightarrow y.p := true], R) \\ = & \quad \{ \text{definitions of } wlp \text{ and } R \} \\ & x.q \vee \neg y.q \\ = & \quad \{ \text{definition of } Q \} \\ & Q \\ = & \quad \{ Q \text{ is universally valid} \} \\ & true , \end{aligned}$$

hence R is universally valid.

(End of Proof 0.)

We could comment on details in the above presentation, but we refrain from doing so. The main observation is that the above proof is completely "bottom-up".

Proof 1

This is a "top-down" proof, which means that the very first thing we do is formalizing the intended behaviour. For reasons of comparison we choose the same thought variables and the same program modifications as in Proof 0. Please, ignore the annotations until further notice.

```
prog.p    {¬ y.p}     x.p := true
        ; { x.p}      [¬ x.q → y.p := true]
        ; { x.p}      CS.p
        ; { x.p}      x.p, y.p := false, false
          {¬y.p} .
```

Now, our proof obligation is to show the invariance of

$$R: \quad \neg y.p \vee \neg y.q \ .$$

To that end we observe that only an assignment $y := true$ can falsify R. Therefore we investigate for one such assignment the condition

$$wlp.([\neg x.p \rightarrow y.q := true], R).$$

By the definitions of wlp and R, we can rewrite it as

$$x.p \vee \neg y.p \ .$$

The variables mentioned in the latter condition can be modified by program p only. As a consequence the condition is satisfied if it is an invariant of program p in isolation. And this is so on account of the obviously correct annotation, which we are allowed to consider now.

(End of Proof 1.)

Proof 1 is shorter than Proof 0, but that need not surprise us, because the former is demand-driven. For the same reason Proof 1 is also less baroque in that it does not mention the quite overspecific Pi's of Proof 0. This need not surprise us either. But what is more important is that Proof 1 makes a less heavier use of the Gries-Owicki theory. In Proof 0 the Pi's were introduced and there we had to verify for all of them that the other program did not falsify them. In Proof 1 this proof obligation was dispensed with in one single sweep, viz. at the emergence of the expression $x.p \vee \neg y.p$. From that moment onwards we were only concerned with providing locally correct annotation. (And it is even questionable whether for that purpose we should have provided all three repetitious assertions $x.p$.)

Proof 1 also shows quite clearly how the Safe Sluice could have been derived, using the Gries-Owicki theory. Had we done so, we would have introduced the y's, we would have taken R as the functional specification, and we would have used the following program fragment, containing

the unknown guard B, as a first approximation for program p:

$$[B \rightarrow y.p := \text{true}] \; ; \; CS.p \; .$$

Then we would have computed $wlp.([B \rightarrow y.p := \text{true}], R)$ and tried to derive
We will not pursue this here.

Proof 2

This is a very short proof. Here it is, out of the magic hat. Introduce thought variable z, and modify the programs as follows:

prog.p	$x.p := \text{true}$
;	$[\neg x.q \rightarrow z := p]$
; $\{x.p\}\{z=p\}$	$CS.p$
;	$x.p := \text{false}$

(Conjunction of assertions is denoted by their juxtaposition.)

First, the annotation is effective since from it we can conclude

$$\begin{array}{ll} & prog.p \text{ in } CS.p \;\wedge\; prog.q \text{ in } CS.q \\ \Rightarrow & \quad \{\text{by the annotation}\} \\ & z=p \;\wedge\; z=q \\ \Rightarrow & \quad \{\text{calculus}\} \\ & p=q \; . \end{array}$$

Second, the annotation is correct because its local correctness is obvious and its global correctnes follows from -- the assignment $z := q$ in $prog.q$ being the only one that could falsify $z=p$ --

$$
\begin{array}{ll}
 & wlp.([\neg x.p \rightarrow z := q],\ x.p \wedge z = p) \\
= & \quad \{\text{definition of } wlp\} \\
 & x.p \vee (x.p \wedge p = q) \\
= & \quad \{\text{pred. calc.}\} \\
 & x.p \\
\Leftarrow & \quad \{\text{assertion to be justified}\} \\
 & x.p \wedge z = p\ .
\end{array}
$$

(End of Proof 2.)

The above proof is presented as if it was discovered, but in fact it was the outcome of the very first effort to derive the Safe Sluice from one of its functional specifications. Because this note is not on the derivation of little multiprograms, we leave it at the remark.

I have to admit that I was quite excited when the above proof emerged. The excitement had, besides the proof being short, at least two grounds.

Firstly, if someone were to guess from an operational understanding of the Safe Sluice which statement of the program is the most dangerous one as far as the safety is concerned, his choice would have been (in fact, it was) an assignment $x := false$, because -- after all -- it is precisely that statement which "opens the sluice", makes a guard true. Now please observe that in the above proof the statements $x := false$ do not even enter the picture: for the assertion $\{x.p\}$ to be correct, it only mattered that in program p it was preceded by $x.p := true$ and that

program q did not modify x.p . Isn't that nicely counterintuitive ? But there is more to it: the relative freedom we have in scattering around the program text statements like x := false, without invalidating the safety proof, is precisely the freedo we still need for ensuring computational progress. (I have not worked out this in any systematic manner yet.)

The second source of excitement with the above proof was that I could not think of any other useful assertion to be plugged in into the program text. Nevertheless the Gries-Owicki theory prescribes it, unnecessary so. My guess is that this is the resu of presenting the theory as a theory for the construction of aposteriori proofs, giving much less opportunity to encounter useful separations of concerns, as is the case with top-down designs. My guess is also that when a true addict of the pure Gries-Owicki theory would provide the "missing" assertions in the above example and when he would give a full-fledged proof, not only would the proof become longer, but --worse-- repetitious.

Proof 3

This is an ugly proof in the way it evolves, but it has a happy end. It is ugly in that it is, by now, much too complicated a proof for the Safe Sluice. The happy end is that Peterson's Algorithm emerges from it.

We begin with plugging in a single assertion, viz. the precondition of the critical section.

```
prog.p:        x.p := true
    ;          [¬x.q → skip]
    ; {R.p.q} CS.p
    ;          x.p := false
```

Now we wish to find an R, as weak as possible, satisfying

(0) $\quad R.p.q \wedge R.q.p \Rightarrow p = q$.

(This is one way of formally stating that the two programs are not in their critical section simultaneously.)
Our first approximation for $R.p.q$ is what can be justified by program p in isolation, viz. $x.p \wedge \neg x.q$. This is no good in cooperation, because program q falsifies the second conjunct by the assignment $x.q := true$. Therefore we weaken that conjunct and our next proposal for $R.p.q$ is

$R.p.q: \quad x.p \wedge (\neg x.q \vee H.p.q)$.

This choice is correct whenever
a) (0) is satisfied
b) $x.q := true$ truthifies $H.p.q$.

Ad a)

$$
\begin{aligned}
& R.p.q \wedge R.q.p \\
= & \quad \{ \text{definition of } R \} \\
& x.p \wedge (\neg x.q \vee H.p.q) \wedge x.q \wedge (\neg x.p \vee H.q.p) \\
= & \quad \{ \text{predicate calculus} \} \\
& x.p \wedge x.q \wedge H.p.q \wedge H.q.p \\
\Rightarrow & \quad \{ \text{on account of (1) below} \} \\
& p = q \ .
\end{aligned}
$$

(1) $\quad H.p.q \wedge H.q.p \Rightarrow p = q$.

Remark When comparing (0) and (1) it seems as if we have put the cart before the horse, but this is not the case: R occurs as an assertion in the program text whereas H doesn't. (End of Remark.)

About the simplest choice for H, such that it satisfies (1), is

$H.p.q : \quad h = q$,

for a fresh variable h.

Ad b) In order to truthify $H.p.q$ within $x.q := true$, we replace the statement with $x.q, h := true, q$.

This, as such, completes this correctness proof for the Safe Sluice. But there is a little bit more to it. Let us summarize the (fully!) annotated text:

prog.p :	$x.p, h := true, p$
;	$[\neg x.q \rightarrow skip]$
; $\{x.p\}\ \{\neg x.q \vee h=q\}$	CS.p
;	$x.p := false$.

Now we observe that, without changing the annotation and its correctness, and, therefore, without changing the validity of the correctness argument, we can weaken the guard $\neg x.q$ towards $\neg x.q \vee h=q$. The price we pay for this change is that h flips from thought variable to program variable, but by this investment the algorithm no longer suffers from the danger of deadlock or individual starvation (without proof here). In fact, we have arrived at Peterson's mutual

exclusion algorithm.

(End of Proof 3.)

The above proof has probably about one and a half moral. The half moral is hidden in the Remark, where it is stated that R is an assertion and H isn't. An assertion has to be established locally and it has to be maintained globally, and this might induce more proof obligations than just establishing a condition like H. The trick we applied was "delegating" the requirement of assertion R satisfying (0) to the non-assertion H. Whether the trick is of any use remains to be seen. The firm moral is that we have indicated the presumably simplest and safest way to weaken a guard, being a possibility we barely need in order to abandon deadlock from programs derived by using the Gries-Owicki theory.

Austin,
30 September 1987

W.H.J. Feijen,
Department of Computer Sciences,
The University of Texas at Austin
Austin, TX 78712 - 1188