Formal Derivation of an Algorithm for Distributed Phase Synchronization

Daniela Buhăceanu.
WHJ Feijen

Abstract

The main purpose of this note is to create more evidence for the observation that parallel programs, distributed or not, can be formally – and economically – derived by means of just the theory of Owicki and Gries, and the predicate calculus. The example selected here is the problem of Phase Synchronization, in which a number of programs each are to pass through an unlimited number of phases in a more or less synchronous fashion. A solution is developed for the special case where the programs are located in the nodes of a tree and the communication facilities are restricted to communication with neighbouring nodes

Keywords and Phrases

Program Derivation, Multiprogramming, Multibounds, Theory of Owicki and Gries, Predicate Calculus, Design Heuristics, Distributed Algorithms, Phase Synchronization

The most common valuation of the theory of Owicki and Gries is that it would, at best, be suited for a posteriori verification of parallel programs [AO91], and even of only those that cooperate via shared variables. Fortunately, this limited view on the potential of the theory is, by now, becoming less and less tenable, because, by its very simplicity, the theory can pretty well be used for the formal derivation of parallel programs

Over the past few years, quite some evidence has been collected that armed with just the theory of Owicki and Gries and the predicate calculus, one can not only just derive parallel programs from their specifications, but moreover — and this is quite important — that it can be done in a fairly economical way. One of the reasons for us to write this note is to show how this works. Also, we have chosen to develop a distributed algorithm in order to demonstrate that the theory of Owicki and Gries is instrumental outside the realm of shared memory algorithms too Of course, one single example cannot exhibit the general mode of deriving parallel programs — multiprograms we call them — but it can give some of the flavour Quite reasonable, and more varied, accounts, can be found in [Moe93] and [vdSom94], and in [Fe90] and [FG95]

The theory of Owicki and Gries, in brief

In explaining the theory of Owicki and Gries, we confine ourselves to what is needed for the

understanding of this note For more detailed explanations we refer to [OG 76] or [Dij 82].

A multiprogram is a set of ordinary sequential programs, which we call the (multiprogram's) components. The multiprogram as a whole has a precondition, and the components may be annotated the way we are used to for sequential programs. The theory of Owicki and Gries tells us that this annotation is correct whenever for each assertion in each component it holds that

(i) it is "locally correct" with respect to the component in which it occurs, i.e it is established by the (dynamically) preceding atomic statement of that component

(ii) it is "globally correct" with respect to the rest of the system, i.e. it is not falsified by any atomic statement of any other component. (This is usually called "interference freedom".)

And, in principle, this is all there is to it. Of course, it ought to be clear which statements are atomic. In our forthcoming example, we will deem assignment statements and so-called guarded skips — statements of the form
if B → skip fi — to be atomic.

Another important notion in multiprogramming is the notion of a system invariant. A relation is a system invariant if and only if it is implied by the multiprogram's precondition

and is not falsified by any atomic statement of any component. As a result, an invariant can be added as a conjunct to each assertion, and therefore we can afford the freedom of writing it nowhere in the annotation. System invariants greatly contribute to the economy of proving and designing, and when used in the way indicated above also to the clarity of exposition and the economy of writing, ... and reading.

Finally, we point out that the guarded skip is our main tool for achieving synchronization. Dynamically, guarded skip **if** $B \rightarrow$ skip **fi** is equivalent to **do** $\neg B \rightarrow$ skip **od**. We handle it via a proof rule, which in Hoare-triple semantics for partial correctness reads

$$\{B \Rightarrow R\} \ \textbf{if}\ B \rightarrow \text{skip}\ \textbf{fi}\ \{R\}$$

Its most frequent application is

$$\textbf{if}\ B \rightarrow \text{skip}\ \textbf{fi}\ \{B\},$$

for establishing the local correctness of assertion B.

Total Deadlock and the Multibound

Apart from our concern for partial correctness of a multiprogram, which we capture by providing correct annotation, we are also confronted with the totally different problem of "individual progress". A component can become blocked indefinitely, when it is engaged in the execution of a guarded skip, the guard of which never becomes stably true due to the operations of the rest of the system. Showing individual progress is, in general, quite a nasty

task, so nasty even that computing science has not yet succeeded in solving this problem in a technically satisfactory manner.

Under some special circumstances however, the problem of individual progress can be tackled gracefully. Consider, for example, the following three-component multiprogram, which, projected on the variables x, y, and z, has the form

$$*[\,x := x+1\,] \quad , \quad *[\,y := y+1\,] \quad , \quad *[\,z := z+1\,]$$

Now suppose that this multiprogram maintains as a system invariant

$$\begin{aligned} MB: \quad & x \leq y + K \\ & \wedge \; y \leq z + L \\ & \wedge \; z \leq x + M \,, \end{aligned}$$

for some constants K, L, and M. We then observe that if, for one reason or another, one of the components comes to a definitive halt, so will the others. As a result, the multiprogram as a whole can display just two scenarios as far as progress is concerned, to wit

either all components get stuck forever
— this is called "total deadlock" —

or each individual component makes progress

So, in the presence of a "multibound" like MB, individual progress can be demonstrated by showing the absence of the danger of total deadlock. And the nice thing about this is that the latter can be done by just the theory of Owicki and Gries

The way we typically prove the absence of the danger of total deadlock is as follows. We lay down a correct preassertion to each guarded skip in each component, and we then show that for each combination of guarded skips — taking one from each component — the disjunction of the guards is implied by the conjunction of the corresponding preassertions — cf [Hoog86] — .

The problem of Phase Synchronization

Specification

The problem of phase synchronization, which we learned from J. Misra [M90], [M91], is as follows. We consider an arbitrary, nonempty set of component programs of the form

Comp.p: $*[S.p]$.

We assume that for each p, all invocations of $S.p$ terminate. The successive $S.p$'s are the successive phases of Comp.p. The problem now is to synchronize the components in such a way that, when a component is about to start the execution of its $(n+1)^{st}$ phase, all other components have completed at least n of their phases. In order to render this synchronization requirement more formally and more precisely, we introduce a fresh variable $x.p$, for each component Comp.p, that keeps track of the number of phases Comp.p has completed. Our first version of the multiprogram thus gets the form

Pre: $\langle \forall q :: x.q = 0 \rangle$
Comp.p: $*[\ \{ \langle \forall q :: x.p \leq x.q \rangle \ .\ ? \}\ S.p$ $;\ x.p := 1 + x.p$ $]$

Version 0

This, now, is our formal specification. The task ahead of us is to superimpose on the above multiprogram additional code so as to achieve that the plugged-in assertion $\langle \forall q :: x.p \leq x.q \rangle$ be a correct precondition to $S.p$. We use the symbol ? to explicitly indicate that this remains to be done.

* * *

Before embarking on a solution, we observe that, no matter how we proceed in establishing target assertion $\langle \forall q :: x.p \leq x.q \rangle$, relation

MB: $\langle \forall p :: \langle \forall q :: x.p \leq 1 + x.q \rangle \rangle$

will be a system invariant. But this is a perfect multibound. Consequently, to show individual progress in our ultimate solution it suffices to show the absence of total deadlock.

Towards a distributed solution

In version 0 above, the local correctness of target assertion $\langle \forall q :: x.p \leq x.q \rangle$ is most easily established by a guarded skip with that assertion as a guard. However, this would require

facilities for direct information exchange between any pair of components. Our aim, though, is to develop an algorithm for the case the components form the nodes of a tree and can only communicate with the components in the neighbouring nodes. As a first step towards such a distributed algorithm, we rewrite our target assertion into the equivalent

$$x.p \leq \langle \downarrow q :: x.q \rangle \quad ,$$

and then decide to nominate one fixed component $\mathcal{R}$ — to reside in the root of the tree — to keep track of the minimal x-value, i.e. maintain

$$x.\mathcal{R} = \langle \downarrow q :: x.q \rangle \quad .$$

This allows us to rewrite our target assertion into the equivalent

$$x.p \leq x.\mathcal{R}$$

In order to ensure that the minimal x-value will indeed be in the root, we decide that each path from a node towards the root will exhibit a descending sequence of x-values. (Thus the x-values form what could be called a down-heap.) More precisely, with $f.q$ denoting the father node of node q (for $q \neq \mathcal{R}$), we decide to maintain as a system invariant

$$P0: \quad \langle \forall q : q \neq \mathcal{R} : x.(f.q) \leq x.q \rangle$$

Thus, the next version of our multiprogram becomes

†: $\downarrow$ denotes the minimum

Pre:	$\langle \forall q :: x.q = 0 \rangle$
Inv:	MB. ♡ P0 . ?
Comp p:	$*[\ \{x.p \leq x.R,\ ?\}\ S.p$ $;\ x.p := 1 + x.p$ $]$

Version 1

The core of the design

In version 1 above, the local correctness of assertion $x.p \leq x.R$ could again be established easily by a guarded skip with that assertion as a guard. Although the situation has improved compared to what we had in our previous version, such a guarded skip would still require facilities for direct information exchange between the root R and any other component. So we must find a way to guarantee the correctness of $x.p \leq x.R$ in a different way. One opportunity is created by exploiting the transitivity of $\leq$, viz. by replacing the target assertion with the stronger

$$x.p \leq \text{"something"} \ \wedge\ \text{"something"} \leq x.R$$

If this is going to work out well, the "something" ought to be an expression that depends on information in Comp.p and its immediate neighbourhood only. In that case the first conjunct becomes a suitable candidate for a guard, and the second conjunct — and there will be no cure! —

had better follow from a system invariant. (In passing, we wish to mention that this strategy is quite general, when multiprogramming.) Unfortunately, the conjunct "something" $\leq x.R$ cannot follow from one of our current system invariants MB or P0, because neither grant us an inequality with $x.R$ at the greater side and a "something" of the desired shape at the smaller side. Therefore, an essentially new ingredient has to enter the game. (So much for these heuristic considerations.)

That essentially new ingredient consist of yet another set of fresh variables, one per component. They serve to eliminate our target assertion $x.p \leq x.R$ in the following way:

$$x.p \leq x.R$$

$\Leftarrow$ { for the "something" we choose the fresh variable $y.p$ }

$$x.p \leq y.p \;\wedge\; y.p \leq x.R$$

$\Leftarrow$ { the first conjunct will be our new target assertion, but the second one is as cumbersome as the old $x.p \leq x.R$. We therefore proceed separating the y's and the x's more rigorously. }

$$x.p \leq y.p \;\wedge\; y.p \leq y.R \;\wedge\; y.R \leq x.R \quad .$$

Here we decide that the last two conjuncts will follow from new system invariants. The invariance of the middle conjunct requires that the maximal y-value resides in the root, and we will meet this

requirement by seeing to it that each path from a node to the root will exhibit an ascending sequence of y-values. (Thus, the y-values form what could be called an up-heap.) More precisely, we shall maintain as a system invariant

P_1: $\langle \forall q : q \neq R : y.q \leq y.(f.q) \rangle$.

Adoption of the third conjunct $-y.R \leq x.R-$ yields, in combination with P_0 and P_1, the stronger invariant

P_2: $\langle \forall q :: y.q \leq x.q \rangle$

In view of our target assertion $x.p < y.p$, it will be necessary to increase y's, and in our next version we immediately do justice to this, and to P_2, by plugging in assignments to y so that P_2 is satisfied†. We obtain

Pre: $\langle \forall q :: x.q = 0 \rangle \wedge \langle \forall q :: y.q = 0 \rangle$
Inv: MB, ♡ P_0: $\langle \forall q : q \neq R : x.(f.q) \leq x.q \rangle$, ? P_1: $\langle \forall q : q \neq R : y.q \leq y.(f.q) \rangle$, ? P_2: $\langle \forall q :: y.q \leq x.q \rangle$, ♡
Comp.p: $*[\{ x.p \leq y.p, ♡ \}$ S.p ; $x.p := 1 + x.p$; $y.p := 1 + y.p$ $]$

Version 2

† : It is the order in which x.p and y.p are increased that caters for P_2

Meanwhile, the correctness of our new target assertion — $x.p \leq y.p$ — has been established by construction. What remains is our care for the invariances of $P0$ and $P1$.

Solution

As for $P0$, increments of $x.p$ in $Comp.p$ can violate it. The weakest precondition for $x.p := 1 + x.p$ not to violate $P0$ is

$$\langle \forall q : q \neq R \wedge p = f.q : 1 + x.p \leq x.q \rangle ,$$

which — since $p = f.q \Rightarrow q \neq R$ — simplifies to

$$\langle \forall q : p = f.q : 1 + x.p \leq x.q \rangle .$$

We plug this in as a preassertion to $x.p := 1 + x.p$. Its global correctness is for free because other components only increase their x-values. Its local correctness will be established by a guarded skip. Notice that the evaluation of this condition by $Comp.p$ requires communication with $Comp.p$'s children only. Also notice that for leaf components, this condition simplifies to true so that here the corresponding guarded skips can be left out altogether.

As for $P1$, increments of $y.p$ in $Comp.p$ can violate it. The weakest precondition for $y.p := 1 + y.p$ not to violate $P1$ is

$$\langle \forall q : q \neq R \wedge p = q : 1 + y.q \leq y.(f.q) \rangle ,$$

which for $p \neq R$ simplifies to

$$1 + y.p \leq y.(f.p)$$

and for $p = R$ to true.
Again its global correctness is for free, and its local correctness will be established by a guarded skip. We thus arrive at our next and almost final version — please ignore, for the time being, the added assertions —

Pre: $\langle \forall q :: x.q = 0 \rangle \wedge \langle \forall q :: y.q = 0 \rangle$
Inv: MB, P_0, P_1, P_2, all $\heartsuit$

Comp p ($p \neq R$ and p not a leaf):

```
*[ S.p
 ; {x.p = y.p, ♡}
   if ⟨∀q : p = f.q : 1 + x.p ≤ x.q⟩ → skip fi
 ; x.p := 1 + x.p
 ; {x.p = 1 + y.p, ♡}
   if 1 + y.p ≤ y.(f.p) → skip fi
 ; y.p := 1 + y.p
]
```

Comp p (p is a leaf):

```
*[ S.p
 ; x.p := 1 + x.p
 ; {x.p = 1 + y.p, ♡}
   if 1 + y.p ≤ y.(f.p) → skip fi
 ; y.p := 1 + y.p
]
```

Comp p ($p = R$):

```
*[ S.p
 ; {x.p = y.p, ♡}
   if ⟨∀q : p = f.q : 1 + x.p ≤ x.q⟩ → skip fi
 ; x.p := 1 + x.p
 ; y.p := 1 + y.p
]
```

Version 3

And we are through, be it that we still have to demonstrate the absence of the danger of total deadlock. The assertions we added have been included for that purpose.

Absence of Total Deadlock

We show the absence of the danger of total deadlock by assuming that each component is stuck in an if-statement, and then derive the validity of false. Most components have two if-statements; the leaves and the root have just one. We now colour the components either black or white by the convention that a component is

black — when stuck in its first if-statement (absent at the leaves), and

white — when stuck in its second if-statement (absent at the root).

Thus, the root is black and all leaves are white. As a consequence, there exists a black component with white children only — nice, little proof omitted — Let p be such a black component. Because it is blocked in its first if-statement, we have

(i) $\quad \langle\, \exists q : p = f.q : x.p \geq x.q \,\rangle \;\wedge\; x.p = y.p$

(The first conjunct is the negation of the guard and the second conjunct is the preassertion we added to this if-statement.)

Because all of p's children are white, hence

blocked in their second if-statements, we also have

(ii) $\langle \forall q : p = f.q : y.q \geq y.(f.q) \;\wedge\; x.q = 1 + y.q \rangle$

(The conjunct $x.q = 1 + y.q$ is the preassertion we added to this if-statement.)

Now let q be a witness for (i), i.e.

(iii) $p = f.q \;\wedge\; x.p \geq x.q \;\wedge\; x.p = y.p$.

By instantiating (ii) for this q, we find

(iv) $y.q \geq y.(f.q) \;\wedge\; x.q = 1 + y.q$.

We now derive the validity of false, as follows

$$
\begin{array}{ll}
 & x.p \\
\geq & \quad \{ (iii) \} \\
 & x.q \\
= & \quad \{ (iv) \} \\
 & 1 + y.q \\
\geq & \quad \{ (iv) \} \\
 & 1 + y.(f.q) \\
= & \quad \{ (iii) \} \\
 & 1 + y.p \\
= & \quad \{ (iii) \} \\
 & 1 + x.p \quad .
\end{array}
$$

So much for the absence of total deadlock. Thanks to the presence of multibound MB, individual progress is guaranteed as well.

A final transformation

In principle, our development is done. Yet, we wish to address two more issues that could be relevant in practice. One is the rather coarse-grainedness of, in particular, the atomic guards $\langle \forall q : p = f.q : 1 + x.p \leq x.q \rangle$, and the other is the problem of the ever growing integers x and y.

The coarse-grained guards are eliminated as follows. Consider a multiprogram in which one of the components contains guarded skip
if $B \wedge C \rightarrow$ skip fi. Now, there is a (rather unknown) theorem — The Guard Conjunction Lemma — stating that for globally correct B this coarse-grained guarded skip can be replaced with the succession
if $B \rightarrow$ skip fi ; if $C \rightarrow$ skip fi of these two finer-grained guarded skips, without impairing the correctness of the multiprogram. The tedious proof of this (important) lemma is not given here [Hoom93]. For our current example, in which each individual conjunct $1 + x.p \leq x.q$ is globally correct, this means that the conjuncts can be evaluated one by one, and in any order.

We deal with the ever growing integers by eliminating them. As for the elimination of x's, we observe that thanks to the invariance of

MB: $\langle \forall p, q :: x.p \leq 1 + x.q \rangle$

two arbitrary x-values differ by at most 1.
We can therefore perform a coordinate transformation into the boolean domain, viz. we introduce

booleans c , one per component, such that

$$c.p \equiv c.q \quad \equiv \quad x.p = x.q \quad , \quad \text{and hence}$$

$$c.p \not\equiv c.q \quad \equiv \quad x.p \neq x.q \quad .$$

We can then translate our program as follows

$$1 + x.p \lesssim x.q \quad \leftrightarrow \quad c.p \not\equiv c.q$$
$$x.p := 1 + x.p \quad \leftrightarrow \quad c.p := \neg c.p \quad .$$

The elimination of the y's is pretty much the same, because we can prove

$$\langle \forall p,q :: \; y.p \lesssim 1 + y.q \rangle \; ,$$

as follows:

$$\begin{array}{ll} & y.p \\ \lesssim & \quad \{P_1\} \\ & y.R \\ \lesssim & \quad \{P_2\} \\ & x.R \\ \lesssim & \quad \{P_0\} \\ & x.q \\ \lesssim & \quad \{\text{structure of } Comp.q\} \\ & 1 + y.q \; . \end{array}$$

We introduce booleans d to replace y, and our final program — of which we now just give the raw code — is

Pre : $\langle \forall p,q :: c.p \equiv c.q \rangle \wedge \langle \forall p,q :: d.p \equiv d.q \rangle$

Comp.p ($p \neq R$ and p not a leaf) :

```
*[ S.p
 ; if ⟨∀q : p=f.q : c.p ≠ c.q⟩ → skip fi
 ; c.p := ¬c.p
 ; if d.p ≠ d.(f.p) → skip fi
 ; dp := ¬d.p
 ]
```

Comp p (p is a leaf) :

```
*[ S.p
 ; c.p := ¬c.p
 ; if d.p ≠ d.(f.p) → skip fi
 ; d.p := ¬d.p
 ]
```

Comp p ($p = R$)

```
*[ S.p
 ; if ⟨∀q : p=f.q : c.p ≠ c.q⟩ → skip fi
 ; c.p := ¬c.p
 ; d.p := ¬d.p
 ]
```

Final Remarks

It is quite probable that the algorithm we developed is well-known. After the above was done, we even recognized that the algorithm could even have been invented by purely operational considerations. After all, one can view it as a huge two-stage handshake protocol. In the first stage the leaf nodes, asynchronously, send completion signals towards the root As soon as this wave of completion signals has reached the root, the latter reflects it as a permission signal which then scatters through the tree giving nodes permission to enter their next phase. Etcetera. (This "metaphor" makes clear that the algorithm is even an efficient one: in a reasonable tree, the lengths of the communication lines between the leaves and the root is negligable compared to the number of components in the game.)

But even if the algorithm were invented along such lines, the task of proving its correctness would still remain. And this could be extremely difficult. At best, one would reverse the course of a development, but it is far more likely that an a posteriori proof will enter more cumbersome alleys, if not dead ones. Deriving programs from their specifications is just much more economic and satisfactory. This is so because program derivation is a very goal-directed activity, in which the program and its correctness proof see the light in a harmonious fashion.

The main purpose of this note was to drive home

the message that the derivation of not too trivial an algorithm like the Distributed Phase Synchronization is very well doable, with the simple theory of Owicki and Gries as our only tool for reasoning about multiprograms.

Acknowledgements

We are grateful to the members of the Eindhoven Tuesday Afternoon Club for their comments on the first draft of this paper. Special thanks go to Rob R. Hoogerwoord and to Netty van Gasteren for their many valuable criticisms.

References

[AO91] Krzysztof R. Apt and Ernst-Rüdiger Olderog, Verification of Sequential and Concurrent Programs. Springer, Berlin, 1991

[Dij82] Edsger W. Dijkstra, A Personal Summary of the Gries - Owicki Theory. In: Selected Writings on Computing: A Personal Perspective Springer, New York, 1982

[Fe90] W.H.J. Feijen, A Little Exercise in Deriving Multiprograms In: W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, Beauty is our Business, A Birthday Salute to Edsger W. Dijkstra, pages 119 - 127

Springer, New York, 1990

[FG95] W.H.J Feijen and A.J.M. van Gasteren,
On Multiprogramming.
Technical Note AvG128/WF218, 1995
Department of Mathematics and
Computing Science, Eindhoven University
of Technology.
(Will also appear in
M. Hinchey and N Dean, editors,
Educational Issues of Formal Methods,
Academic Press, 1996)

[Hoog86] R.R. Hoogerwoord,
An Implementation of Mutual Inclusion,
Information Processing Letters 23 (1986),
pages 77-80

[Hoom93] J. Hooman,
Properties of Program Transformations.
Technical Note, 1993
Department of Mathematics and
Computing Science, Eindhoven University
of Technology

[M90] Jayadev Misra,
Phase Synchronization, 1990
Notes on UNITY: 12-90,
Department of Computer Sciences,
The University of Texas at Austin

[M91] Jayadev Misra,
Phase Synchronization
Information Processing Letters 38 (1991),
pages 101-105

[Moe93] Perry D Moerland,
Exercises in Multiprogramming, 1993
Computing Science Notes 93/07,
Department of Mathematics and
Computing Science, Eindhoven University
of Technology.

[OG76] S. Owicki and D Gries,
An Axiomatic Proof Technique for
Parallel Programs I,
Acta Informatica 6 (1976), pages 319 - 340

[vd Som 94] F.W. van der Sommen,
Multiprogram Derivations, 1994
Master's Thesis,
Department of Mathematics and
Computing Science, Eindhoven University
of Technology.

Eindhoven,
23 February 1996