Phase synchronization for a string of machines

The Eindhoven Tuesday Afternoon Club[1]

dedicated to Prof.Dr. F.E.J. Kruseman Aretz

In this note we present yet another example of how traditionally complicated artefacts like parallel programs can nowadays be developed in a highly systematic manner by means of simple formalisms such as the predicate calculus and the theory of Owicki and Gries. Operational thought is largely – if not completely – abandoned, even in arguing about progress properties.

Moreover, we refute the often heard conjecture that the theory of Owicki and Gries would – at best – be suitable for the a posteriori verification of shared-memory parallel programs, because at this occasion we will use it for the *construction* of a *distributed* algorithm.

* * *

We consider $N+1$ machines, numbered 0 through N. They are arranged in a linear network such that machine i, $1 \leq i \leq N-1$, can communicate with machines $i-1$ and $i+1$ only, machine 0 with just machine 1, and machine N with just machine $N-1$.

Each machine is engaged in a computation proper given by

$$\text{Mach.}i: \quad *[\ S_i\ ]\ ,$$

where S_i is a terminating computation.

In order to express our synchronization requirement we introduce fresh variables x_i and adjust the program texts as follows:

$$\begin{array}{ll} \text{Pre:} & \langle \forall j :: x_j = 0 \rangle \\ \text{Mach.}i: & *[\ \{\ \langle \forall j :: x_i \leq x_j \rangle\ \}\ S_i \\ & \quad ;\ x_i := 1 + x_i \\ & \quad]\ . \end{array}$$

[1] A. Bijlsma, R.W. Bulterman, W.H.J. Feijen, A.J.M. van Gasteren, R.R. Hoogerwoord, C.S. Scholten, F.W. van der Sommen

(It goes without saying that no further changes of x_i are allowed in what follows.)

The requirement now is to superimpose, on the computation proper, an additional algorithm so that the plugged-in assertion

$$Z_i : \quad \langle \forall j :: x_i \leq x_j \rangle$$

be a correct precondition to S_i, for all i.

* * *

Before we embark on a solution, we wish to point out how we will deal with individual progess in examples like the above.

Irrespective of what our ultimate solution will be, relation

$$\text{MB}: \quad \langle \forall i, j :: x_i \leq 1 + x_j \rangle$$

is a system invariant: just apply the axiom of assignment for $x_i := 1 + x_i$ and use precondition Z_i. Relation MB is a so-called multibound [WF195], and in combination with the structure of our machines, its impact is that only two things can happen as far as "liveness" is concerned, viz. in whatever solution we may come up with

either	all machines are guaranteed to make individual progress
or	all machines get stuck (i.e. total deadlock).

So, in what follows we can show individual progress by just showing the absence of total deadlock.

* * *

Now, if there were no restrictions on the communication facilities, a first and simple solution would be

$$\text{Pre}: \quad \langle \forall j :: x_j = 0 \rangle$$

$$\begin{aligned} \text{Mach.}i : \quad & *[\ \underline{\text{if}}\ \langle \forall j :: x_i \leq x_j \rangle \rightarrow \text{skip}\ \underline{\text{fi}} \\ & ;\ \{\ Z_i : \langle \forall j :: x_i \leq x_j \rangle\ \}\ S_i \\ & ;\ x_i := 1 + x_i \\ &]\ . \end{aligned}$$

The local correctness of Z_i follows from the guard, and its global correctness by the Rule of Widening – other machines only increase x_j, thus making Z_i more true than before –.

* * *

The problem with the above solution, however, is the guard which, due to the limited communication facilities, just cannot be evaluated by Mach.i. That guard is equivalent to

(0) $\quad x_i \leq \langle \downarrow j :: x_j \rangle$,

and inspired by the topology of the network, we rewrite it into

$$x_i \leq \langle \downarrow j : j \leq i : x_j \rangle \downarrow \langle \downarrow j : j \geq i : x_j \rangle \ ,$$

and then – introducing variables l_i and r_i – into

(1) $\quad x_i \leq l_i \downarrow r_i$,

hoping that we can maintain

(2a) $\quad l_i = \langle \downarrow j : j \leq i : x_j \rangle \quad$ and

(2b) $\quad r_i = \langle \downarrow j : j \geq i : x_j \rangle$.

Maintaining relations (2) is, however, far too demanding because a simple increase of just one x_j might necessitate an adjustment of many l's and many r's.

Here is the place to remember a very simple, but useful, theorem from the art of multiprogramming, viz. the theorem dubbed Strengthening the Guard. It states that strengthening a guard of an if-statement is harmless to the partial correctness, i.e. the correctness of the annotation. Hence, by this theorem, it suffices that our original guard (0) be *implied* by our proposed guard (1), and this is the case if we can maintain, instead of (2), the weaker

(3a) $\quad l_i \leq \langle \downarrow j : j \leq i : x_j \rangle \quad$ and

(3b) $\quad r_i \leq \langle \downarrow j : j \geq i : x_j \rangle$.

And this is what we shall do.

* * *

In view of the shapes of our guards (1), progress of the multiprogram is best served when the l's and r's are chosen as large as possible. In particular, there is no point in maintaining (3) by decreasing l_i or r_i. This, we will have to bear in mind.

By the shape of the network, there are only few meaningful possibilities for updating l_i. If Mach.i is to keep track of the value for l_i, and Mach.$(i-1)$ for l_{i-1}, it is sweetly reasonable that we rewrite (3a) into

$$l_i \leq \langle \downarrow j : j \leq i-1 : x_j \rangle \downarrow x_i \ , \qquad 1 \leq i \ ,$$

which, by (3a) $(i := i-1)$ is implied by

$$l_i \leq l_{i-1} \downarrow x_i \ ;$$

and this is established by the assignment

$$l_i := l_{i-1} \downarrow x_i \ , \qquad 1 \leq i \ .$$

For Mach.0 we find that

$$l_0 := x_0$$

maintains (3a).

Now, fortunately, if we stick to these assignments to the l's, the l's are never decreased: l_0 does not decrease because x_0 doesn't, and l_i, $1 \leq i$, does not decrease because neither x_i nor – by induction – l_{i-1} does so.

* * *

The next problem is how to superimpose these assignments on our existing multiprogram. For reasons of simplicity, we decide to equip Mach.i with a co-component

$$\text{CL}.i : \qquad *[\ l_i := l_{i-1} \downarrow x_i\ ] \ , \qquad 1 \leq i$$

running in parallel with Mach.i and the rest of the system, and continuously updating l_i. For reasons of symmetry we also introduce a co-component for updating r_i. We thus arrive at the solution depicted in Figure 0.

Pre:	$\langle \forall j :: x_j = 0 \wedge l_j = 0 \wedge r_j = 0 \rangle$
Inv:	$\langle \forall i :: l_i \leq \langle \downarrow j : j \leq i : x_j \rangle$ $\wedge\, r_i \leq \langle \downarrow j : j \geq i : x_j \rangle \rangle$
Mach.i	$(0 \leq i \leq N)$: $* [$ if $x_i \leq l_i \downarrow r_i \rightarrow$ skip fi $; \{ Z_i : \langle \forall j :: x_i \leq x_j \rangle \} S_i ; x_i := 1 + x_i]$
CL.i	$(1 \leq i \leq N)$: $* [l_i := l_{i-1} \downarrow x_i]$
CL.0	$* [l_0 := x_0]$
CR.i	$(0 \leq i \leq N-1)$: $* [r_i := r_{i+1} \downarrow x_i]$
CR.N:	$* [r_n := x_N]$

Figure 0.

* * *

Our remaining obligation is to investigate whether or not this solution exhibits the danger of total deadlock. We shall argue that it does not.

To that end, assume that all Mach.i are stuck in their guarded skips. As a result all x_i are constant. Because the co-components CL.i continue to operate, within a finite number of steps all l_i will be equal to some x-value: l_0 by construction, and l_i, $1 \leq i$, by induction and construction. Symmetrically, all r_i will eventually be equal to some x-value. As a consequence, a Mach.i that holds the minimal x-value then has a stably true guard and can therefore proceed.

* * *

It very much depends on the architecture of the executing machinery whether or not our solution is acceptable. If it is acceptable, that is fine, and if it is not we may further elaborate on it.

Just for the sake of the argument and for showing how we wish to reason about further detailing our solution, we shall indicate how the ever growing integer variables x, l, and r can be eliminated from the program.

First, consider Mach.i with the following annotation

$$
\begin{array}{l}
*[\ \underline{\text{if}}\ x_i \leq l_i \downarrow r_i \rightarrow \text{skip}\ \underline{\text{fi}} \\
\quad ;\ S_i \\
\quad \{\ x_i \leq l_i\ \} \\
\quad ;\ x_i := 1 + x_i \\
\ ]\ .
\end{array}
$$

Assertion $x_i \leq l_i$ is locally correct – from the guard –, and it is globally correct because l_i does not decrease. As a consequence, relation

$$x_i \leq 1 + l_i$$

is an invariant of the system. From (3a), we also have the invariance of

$$l_i \leq x_i\ ,$$

and therefore we have

$$(4) \qquad 0 \leq x_i - l_i \leq 1\ .$$

Hence, the differences $x_i - l_i$ are very small.

From MB – $\langle \forall i, j :: x_i \leq 1 + x_j \rangle$ – we see that the differences between the x-values are very small as well. What about the differences between l-values?

Assignment $l_i := l_{i-1} \downarrow x_i$ in CL.i establishes

$$l_i \leq l_{i-1}\ ,$$

and because l_{i-1} does not decrease, this relation is maintained. So, the l-sequence is descending. Moreover, we observe

$$
\begin{array}{ll}
 & l_{i-1} \\
\leq & \{\ (3a)\ (i := i - 1)\ \}
\end{array}
$$

$$
\begin{array}{ll}
 & x_{i-1} \\
\leq & \{\ \text{MB}\ \} \\
 & 1 + x_i \\
\leq & \{\ (4)\ \} \\
 & 2 + l_i\ ,
\end{array}
$$

and hence we have

(5) $\quad l_i \leq l_{i-1} \leq 2 + l_i\ .$

(That the difference $l_{i-1} - l_i$ can indeed assume the value 2 follows from a simulation, not given here.)

The question now is how we can use (4) and (5) to eliminate the x's and the l's from our program texts.

In view of (4) and the shape of Mach.i, we introduce new variables d_i and e_i, coupled to the old ones by

$$d_i = x_i - l_i \quad \text{and} \quad e_i = x_i - r_i\ ,$$

in terms of which Mach.i can readily be rewritten as

$$
\begin{array}{lll}
\text{Mach.}i\ : & * [\ \underline{\text{if}}\ d_i \leq 0 \wedge e_i \leq 0 \rightarrow \text{skip}\ \underline{\text{fi}} \\
 & \quad ;\ S_i \\
 & \quad ;\ d_i, e_i := 1, 1 \\
 & \qquad (\text{or}\ d_i, e_i := 1 + d_i, 1 + e_i) \\
 &]\ .
\end{array}
$$

(The two-valued d's and e's can be implemented by booleans or by circuitry, if so desired.)

For the co-components the situation is slightly more difficult. Because l_i does not decrease, we can rewrite

$$\text{CL.}i\ : \qquad * [\ l_i := l_{i-1} \downarrow x_i\ ]$$

into the equivalent

$$\begin{array}{ll} \text{CL}.i: & *[\ \underline{\text{if}}\ l_i < l_{i-1} \downarrow x_i \rightarrow l_i := l_{i-1} \downarrow x_i \\ & \quad []\ l_i \geq l_{i-1} \downarrow x_i \rightarrow \text{skip} \\ & \quad \underline{\text{fi}} \\ & \quad]\ , \end{array}$$

and this one into the equivalent

$$\text{CL}.i: \quad *[\ \underline{\text{if}}\ l_i < l_{i-1} \downarrow x_i \rightarrow l_i := l_{i-1} \downarrow x_i\ \underline{\text{fi}}\ ]\ .$$

Now, let us spell out the guard:

$$\begin{array}{ll} & l_i < l_{i-1} \downarrow x_i \\ \equiv & \quad \{\ \text{def.}\ \downarrow\ \} \\ & l_1 < l_{i-1} \wedge l_i < x_i \\ \equiv & \quad \{\ (4)\ \} \\ & l_i < l_{i-1} \wedge 1 + l_i = x_i \qquad (*) \\ \equiv & \quad \{\ (5)\ \}\ \{\ d_i = x_i - l_i\ \} \\ & l_i \neq l_{i-1} \wedge d_i = 1\ . \end{array}$$

From the line marked $(*)$ we conclude that the guard implies $l_{i-1} \downarrow x_i = 1 + l_i$, so that the assignment $l_i := l_{i-1} \downarrow x_i$ can be simplified to $l_i := 1 + l_i$. For the sake of maintaining $d_i = x_i - l_i$, we have to expand it to $l_i, d_i := 1 + l_i, 0$ (or: $l_i, d_i := 1 + l_i, -1 + d_i$).

Furthermore, because the guard is stable – i.e. cannot be falsified by the other components of the multiprogram – there is no need to embed its evaluation and the subsequent assignment into one atomic statement: they can be uncoupled (see [FvdS94]). Thus, we get as our next approximation for CL.i

$$\begin{array}{ll} \text{CL}.i: & *[\ \underline{\text{if}}\ l_i \neq l_{i-1} \wedge d_i = 1 \rightarrow \text{skip}\ \underline{\text{fi}} \\ & \quad ;\ l_i, d_i := 1 + l_i, 0 \\ & \quad]\ . \end{array}$$

And from this we clearly see that, as far as the values of l_i and l_{i-1} are concerned, we are only interested in their being different or not.

In view of this latter observation and of (5), we now introduce new variables λ_i coupled to the old ones by

$$\lambda_i = l_i \ \underline{\text{mod}}\ 3\ .$$

Then, $l_i \neq l_{i-1}$ can be rewritten as $\lambda_i \neq \lambda_{i-1}$ and $l_i := 1 + l_i$ as $\lambda_i := (1 + \lambda_i)$ mod 3.

(The only thing that matters about the 3 is that it is larger than the 2 occurring in (5). It is precisely here where we use the limited range for $l_{i-1} - l_i$.)

Summarizing, we arrive at the algorithm depicted in Figure 1.

Pre:	$\langle \forall j :: d_i = 0 \wedge e_i = 0 \wedge \lambda_i = 0 \wedge \rho_i = 0 \rangle$
Mach.i	$(0 \leq i \leq N)$:
	$*$ [if $d_i \leq 0 \wedge e_i \leq 0 \rightarrow$ skip fi
	; S_i
	; $d_i, e_i := 1, 1$
	]
Cl.i	$(1 \leq i \leq N)$:
	$*$ [if $\lambda_i \neq \lambda_{i-1} \wedge d_i = 1 \rightarrow$ skip fi
	; $\lambda_i, d_i := (1 + \lambda_i)$ mod $3, 0$
	]
CL.0	:
	$*$ [if $d_0 = 1 \rightarrow$ skip fi
	; $\lambda_0, d_0 := (1 + \lambda_0)$ mod $3, 0$
	]
CR.i	$(0 \leq i \leq N - 1)$:
	$*$ [if $\rho_i \neq \rho_{i+1} \wedge e_i = 1 \rightarrow$ skip fi
	; $\rho_i, e_i := (1 + \rho_i)$ mod $3, 0$
	]
CR.N	:
	$*$ [if $e_N = 1 \rightarrow$ skip fi
	; $\rho_N, e_N := (1 + \rho_N)$ mod $3, 0$
	]

Figure 1.

So much for this particular elaboration. Of course, many other variations or deviations are possible, but the point is that we wanted to illustrate how computing science can nowadays

handle programming problems that – not too long ago – were frighteningly complex. It is pleasing to observe that for deriving programs like the one in this note we only need very simple formalisms, while at the same time it is hard to conceive how an operational mind could ever arrive at such solutions.

* * *

We kindly and respectfully offer this design to Prof.Dr. F.E.J. Kruseman Aretz, who always was and still is a dignified professor of computing science, an intellectual of high standing, a modest yet excellent scientist, and a master in teaching and education. It is our departed J.L.A. van de Snepscheut who once said of him: "He is the best teacher I ever had!".

Acknowledgement. We thank R.W. Bulterman for proposing this algorithm for a tree-shaped network.

WF195 W.H.J. Feijen, The multibound, Technical Note, Eindhoven University of Technology, Jan '95.

FvdS94 F.W. van der Sommen, Multiprogram Derivations, Master's Thesis, Eindhoven University of Technology, Oct '94.