Multiprogram Design in the theory of Owicki and Gries

Doug Goldson.
School of ITEE,
University of Queensland, Australia.
goldson@itee.uq.edu.au

October 18, 2001

Abstract

This paper introduces the theory of Owicki and Gries as a method for the design (as opposed to the verification) of multiprograms (concurrent programs).

The theory is applied to a problem of barrier synchronisation for two (and more) programs. The problem is well chosen because it is easy to state yet not easy to solve, and it therefore shows the difficulties of multiprogram design very well.

The effectiveness of the theory to manage multiprogram design, and in the control of complexity, emerges quite well from the exercise.

1 Introduction

This paper applies the theory of Owicki and Gries [7, 2], a theory of partial correctness of multiprograms, to a problem of barrier synchronisation taken from Feijen and van Gasteren [4, 5, 6]. The purpose is to show how well the theory, originally a method of multiprogram verification, can be used as a method of multiprogram design. And this difference in method, between the analytic and the synthetic approach to multiprogramming, characterises the full extent of the development of the theory by Feijen and van Gasteren – an advance based on the effective use of a small number of design heuristics. Section 2 describes the synchronisation problem for two programs, and Section 3 describes its solution. Section 4 then explores generalisations of the problem to more than two programs.

The example is not new. It has already been worked in detail by Feijen and van Gasteren, so the purpose here is not to present novel solutions, (nearly all the programs in the sequel can be found in [6]), but merely to illustrate the use of the theory, to show how well it manages the difficulties of multiprogram design (by isolating freedom of design choice), and to compare proofs with [6] along the way.

By way of preliminaries, a *multiprogram* consists of more than one program to

be executed at the same time. It is written as one (labelled) program above another, like this

P_1: ...
...
P_N: ...

Instead of saying that more than one program is "executing at the same time", it is equivalent, but much simpler, to say that at most one program is executing, but the choice of executing program is not determined by the program text. However, this choice must be *fair*, in the sense that no program is forever prevented from executing.

The programming language is Dijkstra's guarded commands [1], which supports *condition synchronisation* with the blocking conditional IF: **if** $B \rightarrow I$ **fi**, where execution of IF reaches I if B is evaluated when true and execution is blocked if B is evaluated when false. The conditional must itself be *fair*, in the sense that a blocked guard must always sometime be retested.

A fundamental question is what program actions are atomic actions, where an atomic action is one that can not be interrupted during its execution. It is not, in general, realistic to treat assignments and guard evaluations as atomic actions ($x{:=}x+1$ is three actions: load x; add 1; store x). However, it is safe to do so in this paper and, in any case, the luxury of *atomicity synchronisation* $\langle I \rangle$ on *any* instruction I to make it an atomic action will be reserved.

2 A Two Program Synchronisation Problem

For programs A and B

A: $A_1\ A_2$

B: $B_1\ B_2$

it is desired that[1]

(a) (i) A_2 is not started until after B_1 has finished
(ii) B_2 is not started until after A_1 has finished

And this under the constraint that

(b) no changes to A and B are allowed, but for the addition of code X and Y

A: $A_1\ X\ A_2$

B: $B_1\ Y\ B_2$

[1] In A: $A_1\ A_2$, A is a program label, A_1 and A_2 are program schema, and their juxtaposition denotes their sequential composition; a notation that departs from the usual $A_1;A_2$.

Introducing fresh variables x and y into X and Y, the following code achieves a (useless) solution, satisfying (a) at the cost of total deadlock (neither A_2 nor B_2 are reached)

X: $x{:=}false$ $\langle$**if** $x \rightarrow skip$ **fi**$\rangle$

Y: $y{:=}false$ $\langle$**if** $y \rightarrow skip$ **fi**$\rangle$

The problem now becomes that of removing this deadlock while maintaining (a), and the progress (that A_2 and B_2 are sometime reached) is achieved if (1) is satisfied

(1) X terminates $\wedge$ Y terminates

while (a) is maintained if constraint (2) is satisfied

(2) (i) $x{:=}true$ in Y is the only change allowed in Y.
(ii) $y{:=}true$ in X is the only change allowed in X.

where (2i) makes it the job of Y to truthify[2] the guard in X and ensures that (ai) is true.

Requirement (a) is now a consequence of constraints (b) and (2), which makes the code X and Y the sole focus of attention, and requirement (1) now defines the problem to be solved, the cooperative termination of X and Y.

3 A Two Program Solution

This section uses the theory of Owicki and Gries to design a solution to the problem of Section 2. The theory is based on the idea of a correct program annotation, and it is used here constructively, in the manner of Feijen and van Gasteren [6].

An annotation for a sequential program is correct when every assertion in the annotation is valid in Hoare logic. For instruction S and postcondition Q, the triple $\{P\}\ S\ \{Q\}$ is valid when precondition P establishes the weakest liberal precondition of S and Q [1, 3], i.e.

$$P \Rightarrow wlp.S.Q$$

This is sufficient to show that a multiprogram component (a sequential program when treated in isolation) is *locally correct.*

An annotation for a multiprogram is correct when it is locally correct for each component and, furthermore, when no action in one component is capable of falsifying any assertion in another. Satisfaction of this second condition makes the annotation *globally correct.*

****

[2] "Truthify": to make true. The antonym of "falsify".

Satisfaction of (2) is straightforward and the difficulty lies in satisfying (1). To this end, Feijen and van Gasteren [6] show that (1) admits a useful separation of concerns

(1) X terminates $\wedge$ Y terminates
$\equiv$
(3) (X terminates $\vee$ Y terminates) $\wedge$
(4) (X terminates $\equiv$ Y terminates)

and being free to tackle (3) and (4) in either order, (4) is chosen first.

Note Since the purpose is to show how well the theory can control multiprogram design, we draw attention, in advance, to the exploitation of *symmetry* in the design activity. As the problem is symmetric in X and Y, so too should be its solution, meaning that X should be designed in a way that can ignore its image Y.
End

3.1 X terminates $\equiv$ Y terminates

By the symmetry of X and Y, it is sufficient to show that Y terminates if X does, for which, the stable truth of y as X's postassertion is asserted. (Stable means if true then it can not be falsified; i.e., if a stable assertion is true at some control point then it remains true as long as the control point remains active.)

(4) X terminates $\equiv$ Y terminates
$\equiv$ {Symmetry}
Y terminates $\Leftarrow$ X terminates
$\equiv$
$y \wedge (y$ is stable$) \Leftarrow X$ terminates

X: $x{:=}false$ $\langle$**if** $x \rightarrow skip$ **fi**$\rangle$ $\{?\ y\}$

Local correctness of y is readily satisfied in keeping with (2ii).[3]

X: $x{:=}false$ $\langle$**if** $x \rightarrow skip$ **fi**$\rangle$ $y{:=}true$ $\{?\ y\}$

Y: $y{:=}false$ $\langle$**if** $y \rightarrow skip$ **fi**$\rangle$ $x{:=}true$

But global correctness is not satisfied because action $y{:=}false$ in Y can falsify y.

Global correctness is achieved in just two ways: by strengthening the annotation, or by weakening the annotation. In this case, the termination argument means that y can not be weakened, nothing less than y will do, which only leaves strengthening y by solving the equation

[3] In ? y, ? is not a part of the assertion y, it is a comment on the status of y in the program annotation, to the effect that the local correctness of y, or its global correctness, or both, is yet to be proved.

$$Q \wedge y \wedge P \Rightarrow wlp.(y := false).y \wedge P$$

for unknowns P and Q. (The annotation is then modified

X: $\ldots y{:=}true\ \{y\}\{P\}$

Y: $\{Q\}\ y{:=}false \ldots$

.) P and Q are determined by calculation[4]

$$\begin{array}{ll} & wlp.(y := false).y \wedge P \\ \equiv & \\ & false \\ \Leftarrow & \quad \{Q \equiv \neg P\} \\ & Q \wedge y \wedge P \end{array}$$

introducing a new annotation

X: $x{:=}false\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true\ \{y\}\{?\ P\}$

Y: $\{?\ \neg P\}\ y{:=}false\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true$

together with further proof obligations: local and global correctness of P and $\neg P$.

Taking local correctness first. For $\neg P$ in Y, assume

(i) $\neg P$ is a preassertion of A and B, and is true on entry to Y.

(If this seems disconcerting, because of (b), we make it less so shortly.)

For P in X, since nothing is yet known of P, beyond (i), it can only be established by back propagation

X: $x{:=}false\ \{?\ x \Rightarrow P\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ \{P\}\ y{:=}true\ \{y\}\{P\}$

and, since $x \Rightarrow P$ is locally correct, this only leaves the global correctness of P, $\neg P$ and $x \Rightarrow P$.

P and $\neg P$ are made globally correct by assuming

(ii) *orthogonality* of P (i.e., P does not mention x or y).

and the global correctness of $x \Rightarrow P$ under action $x{:=}true$ in Y is given by calculation

$$\begin{array}{ll} & wlp.(x := true).\neg x \vee P \\ \equiv & \quad \{\text{Orthogonality}\} \\ & P \end{array}$$

[4] That y can not contribute to this contradiction explains the choice of $Q \equiv \neg P$; $P \equiv \neg y$ is out as a postassertion of $y{:=}true$, and $Q \equiv \neg y$ is out as a preassertion of $y{:=}false$.

X: $x{:=}false\ \{x \Rightarrow P\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ \{P\}\ y{:=}true\ \{y\}\{P\}$

Y: $\{\neg P\}\ y{:=}false\ \{y \Rightarrow P\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ \{P\}\ x{:=}true$

Finally, (i) and (ii) are satisfied by choosing a *ghost variable* p for P.

X: $x{:=}false\ \{x \Rightarrow p\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ \{p\}\ y{:=}true\ \{y\}\{p\}$

Y: $\{\neg p\}\ y{:=}false\ \{y \Rightarrow p\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ \{p\}\ x{:=}true$

Note A ghost variable is a program variable that *does not affect the control behaviour of the underlying program.* It is a formal device that is used only in the proof that the underlying program is correct. In this case p is a ghost in the strong sense that it does not belong to the program at all, being used only in the annotation. Therefore, it certainly does not affect control behaviour.
End

That p is a ghost explains how $\neg p$ can satisfy (i) as a preassertion of A and B without violating (b) by requiring initialisation code. Being imaginary, p can have whatever initial value we like.

This completes the proof of (4).

3.2 X terminates $\vee$ Y terminates

Since total deadlock can only occur if both X and Y are at their guards, it is sufficient to show the stable truth of $x \vee y$ at the guards.

(3) $\quad X$ terminates $\vee$ Y terminates

$\equiv$

$\quad (x \vee y) \Leftarrow X$ and Y are at their guards

Following [6] this gives the annotation

X: $x{:=}false\ \{?\ Q\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $y{:=}false\ \{R\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true$

under constraint

$$Q \wedge R \Rightarrow x \vee y.$$

Obvious choices for Q are $x \vee y$, x or y, and, given constraint (2ii) on changing X, we can truthify y directly with $y{:=}true$.

X: $x{:=}false\ y{:=}true\ \{?\ y\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $y{:=}false\ x{:=}true\ \{x\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true$

For global correctness of y, in light of $y{:=}false$ in Y, it must be weakened or strengthened. This time strengthening is out[5], which leaves

X: $\quad x{:=}false\ y{:=}true\ \{?\ \ y \vee S\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\quad y{:=}false\ x{:=}true\ \{x \vee T\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

For global correctness of $y \vee S$, assuming

(i) orthogonality of S in y and x.

gives

$$wlp.(y := false).y \vee S \quad \equiv \quad S$$

X: $\quad \{T\}\ x{:=}false\ y{:=}true\ \{y \vee S\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\quad \{?\ \ S\}\ y{:=}false\ x{:=}true\ \{x \vee T\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

and, as before, S is locally correct by assuming

(ii) S is a preassertion of A and B, and is true on entry to Y.

Finally, (i) and (ii) are themselves satisfied by introducing ghost variables s and t.

X: $\quad \{t\}\ x{:=}false\ y{:=}true\ \{y \vee s\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\quad \{s\}\ y{:=}false\ x{:=}true\ \{x \vee t\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

It remains to check that $x \vee y$ is true at the guards, and now we are forced to break the annotation's symmetry by choosing one of s or t to falsify.

$$\begin{array}{ll} & x \vee y \Leftarrow (y \vee s) \wedge (x \vee t) \\ \Leftarrow & \\ & \neg s \vee \neg t \\ \Leftarrow & \qquad \{\text{Break symmetry}\} \\ & \neg s \end{array}$$

X: $\quad x{:=}false\ y{:=}true\ \{y \vee s\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\quad \{s\}\ y{:=}false\ x{:=}true\ \{?\ \neg s\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

The local correctness of $\neg s$ is satisfied by action $\langle s{:=}false\ x{:=}true\rangle$ (where ghost assignment $s{:=}false$ is atomic with $x{:=}true$ and so introduces no new control points).

[5] Strengthening with $y \wedge S$ raises preassertion $\{\neg S\}\ Y$, and so $\{\neg S\}\ X$, but S is true at the X guard, and a ghost assignment in X will falsify $\neg S$ in Y.

X: $x{:=}false\ y{:=}true\ \{?\ y \vee s\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\{s\}\ y{:=}false\ \langle s := false\ x := true\rangle\ \{\neg s\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

Unfortunately, the new action upsets the global correctness of $y \vee s$ because

$$wlp.\langle s := false\ x := true\rangle.y \vee s \quad \equiv \quad y$$

makes y an impossible postassertion of $y{:=}false$. However, weakening $y \vee s$ with x gives

$$wlp.\langle s := false\ x := true\rangle.x \vee y \vee s \quad \equiv \quad true$$

while remaining strong enough for the argument

$$x \vee y \Leftarrow (x \vee y \vee s) \Leftarrow \neg s$$

This completes the proof of (3).

****

Next we compare this proof of (3) to that given by Feijen and van Gasteren in [6], which we do because it illustrates very well the sensitivity of proof structure to the order in which proof obligations are discharged. The weakening of y in the annotation

X: $x{:=}false\ y{:=}true\ \{?\ y \vee S\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $y{:=}false\ x{:=}true\ \{x \vee T\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

introduces a choice of next step. The choice lies between the 'low' level goal to show that $y \vee S$ is globally correct or the 'high' level goal to show the effect of the weakening on truthifying $x \vee y$ at the guards. This time, choosing the latter and introducing a single ghost variable means that we are no longer forced to choose which one of S or T to falsify, and the annotation's symmetry is maintained.

$$\begin{array}{ll} & x \vee y \Leftarrow (y \vee S) \wedge (x \vee T) \\ \Leftarrow & \\ & \neg S \vee \neg T \\ \equiv & \\ & S \wedge T \Rightarrow false \\ \equiv & \quad \{\text{Maintain symmetry}\} \\ & v{=}2 \wedge v{=}1 \Rightarrow 1{=}2 \end{array}$$

X: $x{:=}false\ y{:=}true\ \{?\ y \vee v{=}2\}\ \langle \mathbf{if}\ x \to skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $y{:=}false\ x{:=}true\ \{x \vee v{=}1\}\ \langle \mathbf{if}\ y \to skip\ \mathbf{fi}\rangle\ x{:=}true$

The global correctness of $y \vee v{=}2$ is satisfied by the ghost action $v{:=}2$ truthifying $v{=}2$ when the action $y{:=}false$ falsifies y.

X: $\langle v{:=}1\ x{:=}false\rangle\ y{:=}true\ \{y \vee v{=}2\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true$

Y: $\langle v{:=}2\ y{:=}false\rangle\ x{:=}true\ \{x \vee v{=}1\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true$

This completes the (alternative) proof of (3), taken from [6].

****

Finally, between proving absence of individual deadlock (4) and absence of total deadlock (3) the program code has changed, which means that the proof of (4) must be redone.

The global correctness of $x \Rightarrow p$ in X is again satisfied by making p the pre-assertion of $x{:=}true$ in Y

X: $x{:=}false\ \{x \Rightarrow p\}\ y{:=}true\ \{x \Rightarrow p\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\{p\}\ y{:=}true\ \{y\}\{p\}$

Y: $\{\neg p\}\ y{:=}false\ \{?\ p\}\ x{:=}true\ \{p\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ \{p\}\ x{:=}true$

but this time a ghost assignment is needed for the local correctness of p

X: $x{:=}false\ \{x \Rightarrow p\}\ y{:=}true\ \{x \Rightarrow p\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\{p\}\ y{:=}true\ \{y\}\{p\}$

Y: $\{\neg p\}\ \langle p := true\ y := false\rangle\ \{\ p\}\ x{:=}true\ \{p\}\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ \{p\}\ x{:=}true$

This completes the (revised) proof of (4).

4 A Three Program Synchronisation Problem

As a piece of code documentation, an obvious step is to try to make further use of the proof to solve a three program problem using the same communication pattern. Given are

A: $A_1\ X\ A_2$

B: $B_1\ Y\ B_2$

C: $C_1\ Z\ C_2$

and it is desired that

(a) (i) A_2 is not started until after B_1 and C_1.
(ii) B_2 is not started until after A_1 and C_1.
(iii) C_2 is not started until after A_1 and B_1.

(b) no changes to A, B and C are allowed, but for the addition of code X, Y and Z.

The problem again becomes

X: $x{:=}false\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ \{?\ y\}\ \{?\ z\}$

Y: $y{:=}false\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle$

Z: $z{:=}false\ \langle \mathbf{if}\ z \rightarrow skip\ \mathbf{fi}\rangle$

(1) X,Y and Z all terminate.

(2) Allowable changes are
(i) $x{:=}true$ in Y and Z.
(ii) $y{:=}true$ in X and Z.
(iii) $z{:=}true$ in X and Y.

The annotation ensures barrier synchronisation (a) because it makes y and z true and stable if A_2 is started, so program control must have reached Y and Z.

As before

(1) X,Y and Z all terminate
$\equiv$
(3) (X or Y or Z terminate) $\wedge$
(4) (X terminates $\equiv$ Y terminates $\equiv$ Z terminates)

Let the proof of (4) follow the same pattern as Section 3.1. For the correctness of y and z.

X: $x{:=}false\ \{?\ x \Rightarrow q\}\ \langle \mathbf{if}\ x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true\ z{:=}true\ \{y\}\ \{?\ p\}\ \{z\}\ \{q\}$

Y: $\{\neg p\}\ y{:=}false\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ z{:=}true\ x{:=}true$

Z: $\{\neg q\}\ z{:=}false\ \langle \mathbf{if}\ z \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true\ y{:=}true$

For global correctness of $x \Rightarrow q$

Y: $\{\neg p\}\ y{:=}false\ \langle \mathbf{if}\ y \rightarrow skip\ \mathbf{fi}\rangle\ z{:=}true\ \{?\ q\}\ x{:=}true$

Z: $\{\neg q\}\ z{:=}false\ \{z \Rightarrow q\}\ \langle \mathbf{if}\ z \rightarrow skip\ \mathbf{fi}\rangle\ \{q\}\ x{:=}true\ y{:=}true$

(3) requires assignments *before* the guarded skips. Take component Y. Introducing $x{:=}true$ by (2i) gives preassertion q, at which point we can stop, because the local correctness of q in Y and the global correctness of $\neg q$ in Z can not be satisfied. But introducing $z{:=}true$ by (2iii) also gives preassertion q. There is no further room to manoeuvre, which invites the conclusion that the problem is unsolvable with this communication pattern. And it is, because out of this failure to find a proof arises the counterexample that Y might terminate X with $x{:=}true$ by (2i), and X Y, before Z is even entered.

****

The previous program is an obvious generalisation of the two program case. A different program, taken from [6], is

X: $x{:=}false\ y{:=}false\ \langle \mathbf{if}\ x \wedge y \rightarrow skip\ \mathbf{fi}\rangle\ \{?\ x\}\ \{?\ y\}\ \{?\ z\}$

Y: $y{:=}false\ z{:=}false\ \langle \mathbf{if}\ y \wedge z \rightarrow skip\ \mathbf{fi}\rangle$

Z: $z{:=}false\ x{:=}false\ \langle \mathbf{if}\ z \wedge x \rightarrow skip\ \mathbf{fi}\rangle$

(2) Allowable changes are
(i) $x{:=}true$ in Y.
(ii) $y{:=}true$ in Z.
(iii) $z{:=}true$ in X.

For (4) and the correctness of x,y and z.

X: $x{:=}false\ y{:=}false\ \{x \Rightarrow p\}\ \langle \mathbf{if}\ x \wedge y \rightarrow skip\ \mathbf{fi}\rangle$
$z{:=}true\ \{x\}\ \{p\}\ \{y\}\ \{?\ q\}\ \{z\}$

Y: $\{\neg p\}\ y{:=}false\ z{:=}false\ \langle \mathbf{if}\ y \wedge z \rightarrow skip\ \mathbf{fi}\rangle\ \{?\ p\}\ x{:=}true$

Z: $\{\neg q\}\ z{:=}false\ x{:=}false\ \langle \mathbf{if}\ z \wedge x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true$

(3) requires assignments before the guarded skips

X: $x{:=}false\ y{:=}false\ z{:=}true\ \{x \Rightarrow p\}$
$\langle \mathbf{if}\ x \wedge y \rightarrow skip\ \mathbf{fi}\rangle\ z{:=}true$

Y: $\{\neg p\}\ y{:=}false\ \langle p := true\ z := false\rangle\ \{p\}\ x{:=}true$
$\langle \mathbf{if}\ y \wedge z \rightarrow skip\ \mathbf{fi}\rangle\ \{p\}\ x{:=}true$

For proof of (3)

(3) X or Y or Z terminate
$\equiv$
$(x \wedge y) \vee (y \wedge z) \vee (z \wedge x) \Leftarrow X$ and Y and Z are at their guards

X: $x{:=}false\ y{:=}false\ z{:=}true\ \{x \vee y \vee z \vee s \vee u\}$
$\langle \mathbf{if}\ x \wedge y \rightarrow skip\ \mathbf{fi}\rangle\ z{:=}true$

Y: $\{s\}\ y{:=}false\ z{:=}false\ \langle s := false\ x := true\rangle\ \{\neg s\}$
$\langle \mathbf{if}\ y \wedge z \rightarrow skip\ \mathbf{fi}\rangle\ x{:=}true$

Z: $\{u\}\ z{:=}false\ x{:=}false\ \langle u := false\ y := true\rangle\ \{\neg u\}$
$\langle \mathbf{if}\ z \wedge x \rightarrow skip\ \mathbf{fi}\rangle\ y{:=}true$

But now the annotation is too weak for the argument

$(x \wedge y) \vee (y \wedge z) \vee (z \wedge x) \Leftarrow (x \vee y \vee z \vee s \vee u)$
$\not\Leftarrow$
$\neg s \wedge \neg u$

And out of this failure arises the counterexample that X and Y are at their guards and then Z falsifies both z and x and truthifies just y.

****

Of the two failures, the first is hopeless because it falsifies (4) and allows one component to terminate in a way that might prevent another. The second is less hopeless because (4) can be proved but (3) can not. However, in [6] Feijen and van Gasteren make (3) true by replacing the blocking IF instructions by DO loops. When all of X, Y and Z are at the guards, each component is made to execute its assignment while it is waiting and it only takes two true variables for one component to terminate.

It remains to recheck (4)

X: $x{:=}false\ y{:=}false\ \{Inv{:}x \Rightarrow p\}$
do $\neg(x \wedge y) \rightarrow z := true$ **od** $z{:=}true\ \{x\}\ \{p\}\ \{y\}\ \{?\ q\}\ \{z\}$

Y: $\{\neg p\}\ y{:=}false\ \langle p := true\ z := false\rangle\ \{Inv{:}p\}$
do $\neg(y \wedge z) \rightarrow x := true$ **od** $x{:=}true$

Z: $\{\neg q\}\ z{:=}false\ x{:=}false$
do $\neg(z \wedge x) \rightarrow y := true$ **od** $y{:=}true$

Note While the loop guards in this program are not atomic actions, the program topology shows that they are stable – once true, they remain true.
End

5 Conclusions

Feijen's solution to a simple problem of barrier synchronisation for two programs has been presented. It was used to illustrate how the theory of Owicki and Gries can be used to design multiprograms, as opposed merely to verify them. We saw how design is driven by the need to make program assertions locally correct and then globally correct. This helps to isolate choices and sometimes even to remove them. For instance, we saw how the annotation structure can sometimes determine whether weakening or strengthening must be used for global correctness. We also saw how metavariables can be used to defer design choices, and we saw a good example of how different proofs can be obtained by tackling proof obligations in a different order.

A failure to extend the two program solution to any number of programs was also presented, together with Feijen's solution. In this case we saw how the line between success and failure in multiprogram design is extremely thin, with two programs failing to satisfy (1), but while one is a dead-end, the other eventually leads to a solution. We also saw the theory used here to construct falsifying execution histories from failed attempts at proof.

Finally, while the overall aim of the theory is to control the complexity of design by abstracting from execution histories, we saw how the complexity of proof (in-

troduced by global correctness) is itself controlled quite effectively by exploiting symmetry in the annotation.

References

[1] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[2] E. W. Dijkstra. A personal summary of the gries-owicki theory. In *Selected Writings on Computing: A Personal Perspective.* Springer-Verlag, 1982.

[3] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and program Semantics.* Springer-Verlag, 1990.

[4] W. H. J. Feijen. Phase synchronisation for two machines. In M. Broy, editor, *Programming and Mathematical Method.* Springer-Verlag, 1990.

[5] W. H. J. Feijen and A. J. M. van Gasteren. Programming, proving and calculation. In C. N. Dean and M. G. Hinchey, editors, *Teaching and Learning Formal Methods.* Academic Press, 1996.

[6] W. H. J. Feijen and A. J. M. van Gasteren. *On a Method of Multi-Programming.* Springer-Verlag, 1999.

[7] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Communications of the ACM*, 19(5), 1976.