Strengthen, Widen, Get Semaphores

Javier Blanco[1] and Nicolás Wolovick[1]

Facultad de Matemática, Astronomía y Física,
Universidad Nacional de Córdoba,
Medina Allende y Haya de la Torre, Ciudad Universitaria,
Córdoba, Argentina
`blanco@mate.uncor.edu, nicolasw@famaf.unc.edu.ar`

Abstract. Many parallel programs are expressed in terms of conditional atomic actions with different degrees of atomicity. It is known that these synchronization primitives are expensive to implement in its fully generality [2]. Many platforms provide an efficient set of synchronization primitives. Semaphores are a very special case of conditional atomic actions which can be considered a canonical synchronization primitive. There exist methods to transform general conditional atomic actions into these very special ones. One of the simplest is the *Change of Variables* method introduced in [1,2] and generalized in [10]. We review and improve this method in the context of formal development of multiprograms [4,7].

Keywords: Program transformation; Semaphore synchronization; Multiprograms; Program derivation; Theory of Owicki and Gries; Efficient synchronization primitives.

1 Introduction

It has been effectively shown that the Owicki-Gries theory can be used to formally develop parallel and distributed programs [4,7] in spite of its lack of a uniform way to treat liveness properties. The main reason to choose this theory is its simplicity and the possibility of avoiding operational reasoning.

Programs developed with this methodology synchronize in terms of conditional atomic actions with different degrees of atomicity. One of the aims of this article is to fill in the gap between these abstract programs and implementations in actual computer systems. Since one of the most universal synchronization primitive is the semaphore, we'll use this method to transform abstract multiprograms into concrete ones where all the synchronization is achieved through semaphores. Since semaphore operations are a special case of conditional atomic actions, we don't need to add any new theory to deal with them.

One method to achieve this transformation stated in [1], consists in applying coordinate transformations to conditional atomic actions to make them equivalent to semaphore primitives. The method is simple, but it can be applied to a very limited number of examples, even when the solutions are tailored to the use of semaphore primitives. This method was generalized in [10] allowing the transformation of a larger number of conditional atomic actions, in particular the ones with an empty body.

We review and improve this method in the context of formal development of multiprograms. In particular, finding the new annotations will be done systematically, the partial correctness will be separated from progress requirements achieving a good *separation of concerns* and, since we will only strengthen the annotations, correctness will be preserved by construction.

2 A Brief Summary of Owicki-Gries Theory

For this work we can explain in a nutshell the Owicki-Gries assertional theory to prove parallel programs [8,6]. Given a set of sequential Dijkstra's guarded command language programs [5] called components, the multiprogram as a whole will be correct respect to the components annotations if

- **local correctness**: each assertion is established in the component it occurs.
- **global correctness**: each assertion in one component is not falsified by any other assertion-sentence pair of the other components. It is also known as *interference free test*.

The idea is simple, if we can show that the assertion $\{P\}$ is locally valid, and any $\{Q\}S$ of other component maintains it[1], in an interleaved model of execution where each sentence between assertions is indivisible, the nondeterminism of the scheduler won't affect the strong soundness property [3] of annotated sequential components.

Put it in other words, interference free test protects assertions from nondeterminism and any interleaved model of execution will be valid if it respects the atomicity given by the assertions.

Unless stated otherwise, we will assume each line of a program to be atomic.

Even though the amount of proof obligations for an annotated multiprogram is polynomial on the number of annotations, there are techniques to quickly deal with many simple cases. We show one that is going to be used all through our presentation, for a more detailed account we refer to [7].

Lemma 1 (Widening). *Given a transitive relation $\preceq$ and an expression E where x does not occur in it, if we can show $P \Rightarrow x \preceq f.x$, then the following triple is valid*

$$\{P \wedge E \preceq x\}\ x := f.x\ \{E \preceq x\}$$

In the original work of Owicki, the synchronization and selection constructs were different, we take the unifying approach, interpreting the one-guard selection **if** $B \rightarrow S$ **fi** as a synchronization. The most common form of synchronization is when S is empty, usually called *guarded skip*.

[1] Formally, showing valid the triple $\{P \wedge Q\}S\{P\}$.

There are also some *transformational theorems* for multiprograms, one remarkably useful for our purposes is the following.

Lemma 2 (Guard Strengthening Lemma). *The sentence*

$$\textbf{\textit{if}}\ B \rightarrow \textbf{\textit{skip}}\ \textbf{\textit{fi}}$$

can be replaced by

$$\{C \Rightarrow B\}\ \textbf{\textit{if}}\ C \rightarrow \textbf{\textit{skip}}\ \textbf{\textit{fi}}$$

without impairing the correctness of the annotation, given that $\{C \Rightarrow B\}$ *are locally and globally correct.*

Care must be taken applying this lemma though, because in the process of strengthening progress may be hampered.

If by any means we show $0 < s \Rightarrow B$, then we can proceed into a more semaphore-like synchronization, since semaphore operations can be represented in terms of guarded command language.

$$P.s : \langle \textbf{if}\ 0 < s \rightarrow s := s - 1\ \textbf{fi} \rangle$$
$$V.s : \langle s := s + 1 \rangle$$

The constructor $\langle S \rangle$ explicitly states atomicity, but in most cases we won't use it because annotations and context will make this point clear.

3 Producer-Consumer

Let's consider the classical *Producer/Consumer* problem through a bounded buffer. The component called *producer* produces some elements and sends them to the *consumer* component which will use them. Some synchronization is needed to avoid writing on a full buffer or reading from an empty one. The formal specification or the so-called *computation proper* of this problem can be stated as follows[2].

$Pre : p = 0 \wedge c = 0$	
$Prod : *[\ produce.n$	$Cons : *[\ \{0 < p - c?\}$
$;\ \{p - c < N?\}$	$;\ m := buf.(c\ mod\ N)$
$;\ buf.(p\ mod\ N) := n$	$;\ c := c + 1$
$;\ p := p + 1$	$;\ consume.m$
]	]

The simplest solution for this problem can be obtained using guarded skip statements.

[2] The notation $*[S]$ is a short for $\textbf{do}\ true \rightarrow S\ \textbf{od}$, while the question mark ? recalls the assertion has to be proved and when it's done we change it into a heart suit $\heartsuit$.

$Pre : p = 0 \wedge c = 0$	
$Prod : *[\ produce.n$	$Cons : *[\ \textbf{if}\ 0 < p - c \rightarrow \textbf{skip fi}$
$;\ \textbf{if}\ p - c < N \rightarrow \textbf{skip fi}$	$\{0 < p - c\heartsuit\}$
$\{p - c < N\heartsuit\}$	$;\ m := buf.(c\ mod\ N)$
$;\ buf.(p\ mod\ N) := n$	$;\ c := c + 1$
$;\ p := p + 1$	$;\ consume.m$
$]$	$]$

Local correctness is immediate. Global correctness is ensured by *widening*.

Note that his multiprogram satisfies its *safety properties* by construction. We can show *progress*, using semi-formal arguments together with two safety properties: absence of total deadlock and the existence of a multibound [7] which shows that the progress of a component implies the progress of the other, and hence *individual progress* is reduced to the absence of total deadlock.

We will try to replace the guarded skips with P operations on semaphores. The guard strengthening lemma is powerful enough to solve this example. We begin with the producer postulating a predicate Q as precondition of the guarded skip. We calculate and try to obtain a simple Q which furthermore is as weak as possible.

$$\begin{aligned}
& Q \Rightarrow (0 < s \Rightarrow p - c < N) \\
\equiv\ & \{\ \text{algebra}\ \} \\
& Q \Rightarrow (0 < s \Rightarrow 0 < N - p + c) \\
\Leftarrow\ & \{\ \text{transitivity}\ \} \\
& Q \Rightarrow (s \leq N - p + c)
\end{aligned}$$

The weakest Q that satisfies the last property is precisely $s \leq N - p + c$. Note that this is not the weakest precondition that would allow the application of the guard strengthening lemma. However, one of the main concerns of this method is to keep the annotation as simple as possible. We obtain the following program.

$Pre : p = 0 \wedge c = 0$	
$Prod : *[\ produce.n$	$Cons : *[\ \textbf{if}\ 0 < p - c \rightarrow \textbf{skip fi}$
$\{s \leq N - p + c?\}$	$\{0 < p - c\}$
$;\ \textbf{if}\ p - c < N \rightarrow \textbf{skip fi}$	$;\ m := buf.(c\ mod\ N)$
$\{p - c < N\}$	$;\ c := c + 1$
$;\ buf.(p\ mod\ N) := n$	$;\ consume.m$
$;\ p := p + 1$	$]$
$]$	

The application of the guard strengthening lemma allows to replace the old guard by $0 < s$ preserving the partial correctness. Whereas the question marked assertion is not yet proven, it will be done and the transformation is hence justified.

```
$Pre : p = 0 \wedge c = 0$

$Prod : *[\ produce.n$
            $\{s \leq N - p + c?\}$
          ; **if** $0 < s \rightarrow$ **skip fi**
            $\{p - c < N\}$
          ; $buf.(p\ mod\ N) := n$
          ; $p := p + 1$
          ]

$Cons : *[$ **if** $0 < p - c \rightarrow$ **skip fi**
            $\{0 < p - c\}$
          ; $m := buf.(c\ mod\ N)$
          ; $c := c + 1$
          ; $consume.m$
          ]
```

The next step is to ensure the correctness of the new assertion. Globally it holds by widening. For the local correctness the only choice is to require that it is an invariant for the loop. Initially, any value between 0 and N for s would make the assertion true. We choose $s = N$ as precondition since it is the largest possible value and hence it will help to satisfy the progress requirement.

The new assertion must also hold at the end of the loop. Without modifying the computation proper, the only choice is to decrement s anywhere in the component between (and including) the guarded skip and the p increment. Since we are trying to get a P operation on a semaphore, we replace the skip by the required decrement. Note, however, that for this example a weaker primitive (see for example [9]) would have been enough. The local correctness of the next program is now obvious, and the global correctness of all the assertions holds by widening.

```
$Pre : p = 0 \wedge c = 0 \wedge s = N$

$Prod : *[\ produce.n$
            $\{s \leq N - p + c\}$
          ; **if** $0 < s \rightarrow s := s - 1$ **fi**
            $\{s \leq N - p + c - 1\}$
          ; $buf.(p\ mod\ N) := n$
          ; $p := p + 1$
            $\{s \leq N - p + c\}$
          ]

$Cons : *[$ **if** $0 < p - c \rightarrow$ **skip fi**
            $\{0 < p - c\}$
          ; $m := buf.(c\ mod\ N)$
          ; $c := c + 1$
          ; $consume.m$
          ]
```

Whereas the program is partially correct, it suffers from the danger of deadlock, since the semaphore s is never incremented. In order to satisfy the *ground rule of progress*, the consumer should have the potential of truthifying $0 < s$. The only point in the program where this increment can be safely performed without compromising the assertions' correctness is together with the increment of c.

```
$Pre : p = 0 \wedge c = 0 \wedge s = N$

$Prod : *[\ produce.n$
            $\{s \leq N - p + c\}$
          ; **if** $0 < s \rightarrow s := s - 1$ **fi**
            $\{s \leq N - p + c - 1\}$
          ; $buf.(p\ mod\ N) := n$
          ; $p := p + 1$
            $\{s \leq N - p + c\}$
          ]

$Cons : *[$ **if** $0 < p - c \rightarrow$ **skip fi**
            $\{0 < p - c\}$
          ; $m := buf.(c\ mod\ N)$
          ; $c, s := c + 1, s + 1$
          ; $consume.m$
          ]
```

Similarly, the guard strengthening lemma can be applied to replace the guarded skip in the consumer component. We present the final program with all the annotations and leave to the reader the exercise of reconstructing the calculation. The precondition for the semaphore t is the only possible choice to make the loop invariant initially true.

$Pre : p = 0 \wedge c = 0 \wedge s = N \wedge t = 0$	
$Prod : *[\ produce.n$	$Cons : *[\ \{t \leq p - c\}$
$\{s \leq N - p + c\}$	$\mathbf{if}\ 0 < t \rightarrow t := t - 1\ \mathbf{fi}$
$;\ \mathbf{if}\ 0 < s \rightarrow s := s - 1\ \mathbf{fi}$	$\{t \leq p - c - 1\}$
$\{s \leq N - p + c - 1\}$	$;\ m := buf.(c\ mod\ N)$
$;\ buf.(p\ mod\ N) := n$	$;\ c, s := c + 1, s + 1$
$;\ p, t := p + 1, t + 1$	$\{t \leq p - c\}$
$\{s \leq N - p + c\}$	$;\ consume.m$
$]$	$]$

The solution is correct and its full proof can be easily obtained from the given annotation. However, the grain of atomicity is somewhat coarse for the two multiassignments. The technique to reduce the grain of atomicity is yet another application of the guard strengthening lemma, which became standard in [7]. Two new variables, s' and t' are introduced which will be used to replace s and t respectively. The following invariant should be maintained:

$$0 \leq s' \Rightarrow 0 \leq s \quad \wedge \quad 0 \leq t' \Rightarrow 0 \leq t$$

Again, the easiest way is to require the simpler and stronger invariant

$$s' \leq s \wedge t' \leq t$$

The following annotated program shows that the invariant is preserved if the increment to s' (respectively t') is performed after the one for s (respectively t). The correction of the annotation is immediate.

$Pre : p = 0 \wedge c = 0 \wedge s = N \wedge s' = N \wedge t = 0 \wedge t' = 0$	
$Prod : *[\ produce.n$	$Cons : *[\ \{t \leq p - c\}$
$\{s \leq N - p + c\}$	$\mathbf{if}\ 0 < t \rightarrow t, t' := t - 1, t' - 1\ \mathbf{fi}$
$;\ \mathbf{if}\ 0 < s \rightarrow s, s' := s - 1, s' - 1\ \mathbf{fi}$	$\{t \leq p - c - 1\}$
$\{s \leq N - p + c - 1\}$	$;\ m := buf.(c\ mod\ N)$
$;\ buf.(p\ mod\ N) := n$	$;\ c, s := c + 1, s + 1$
$;\ p, t := p + 1, t + 1$	$\{s' < s\}$
$\{t' < t\}$	$;\ s' := s' + 1$
$;\ t' := t' + 1$	$\{t \leq p - c\}$
$\{s \leq N - p + c\}$	$;\ consume.m$
$]$	$]$
Inv: $s' \leq s \wedge t' \leq t$	

Changing the guards to $0 < s'$ and $0 < t'$, makes $\{s, t\}$ a set of auxiliary variables. Eliminating them and all annotations except for the original, we obtain the following program.

$Pre : p = 0 \wedge c = 0 \wedge s' = N \wedge t' = 0$	
$Prod : *[$ $produce.n$	$Cons : *[$ $\mathbf{if}\ 0 < t' \rightarrow t' := t' - 1\ \mathbf{fi}$
; $\mathbf{if}\ 0 < s' \rightarrow s' := s' - 1\ \mathbf{fi}$	$\{0 < p - c\}$
$\{p - c < N\}$	; $m := buf.(c\ mod\ N)$
; $buf.(p\ mod\ N) := n$	; $c := c + 1$
; $p := p + 1$	; $s' := s' + 1$
; $t' := t' + 1$	; $consume.m$
]	]

Changing back the names of the semaphores, and using P and V operations, we obtain

$Pre : p = 0 \wedge c = 0 \wedge s = N \wedge t = 0$	
$Prod : *[$ $produce.n$	$Cons : *[$ $P.t$
; $P.s$	; $m := buf.(c\ mod\ N)$
; $buf.(p\ mod\ N) := n$	; $c := c + 1$
; $p := p + 1$	; $V.s$
; $V.t$	; $consume.m$
]	]

There's one point left though, in the game of strengthening the guards the possibility of not making progress may have appeared. The still valid multibound together with the absence of total deadlock guarantee the individual progress for producer/consumer with semaphores. To prove no total deadlock some auxiliary variables are needed, this proof is standard so we refer to the literature, for example [2,7].

Some remarks are appropriate. This example was also treated in [10], but the derivation in this article differs in subtle but important points.

The assertions required were stronger in Schneider's work. Instead of the inequalities obtained here, equalities were stated between the value of the semaphore and the expressions in the guards. One may claim that the use of equalities ensure that progress properties are preserved. Whereas this claim is true, the strengthening needed to eliminate the multiple assignment can destroy progress, and hence the proof has to be done again for the stronger annotation. The drawback of the equalities is that partial and total correctness cannot be dealt with separately, since V operations are needed to ensure partial correctness. Moreover, this drawback becomes more apparent with the elimination of multiple assignments. Here, it was just a canonical application of the guard strengthening lemma, whereas in Schneider's derivation a wholly new annotation had to be devised, since the old one was no longer valid. In more complex examples this may be an important improvement.

4 Phase Synchronization for Two Machines

We tackle a different problem now, namely the one that synchronizes two components in order to execute S or T only when the number of times they've been executed is equal. The computation proper can be stated as follows.

```
$Pre : x = 0 \wedge y = 0$
$A : *[$  $\{x \leq y?\}$
          $S$
       ;  $x := x + 1$
       $]$
$B : *[$  $\{y \leq x?\}$
          $T$
       ;  $y := y + 1$
       $]$
$Inv : x \leq y + 1 \wedge y \leq x + 1$
```

The invariant follows directly from the queried assertions and the structure of the components, forming a multibound. Through coordinate transformation $d = x - y + 1$ we step through the next version

```
$Pre : d = 1$
$A : *[$  $\{d \leq 1?\}$
          $S$
       ;  $d := d + 1$
       $]$
$B : *[$  $\{1 \leq d?\}$
          $T$
       ;  $d := d - 1$
       $]$
$Inv : 0 \leq d \leq 2$
```

To ensure local correctness we make them loop invariants, while global correctness holds by widening.

```
$Pre : d = 1$
$A : *[$  $\{d \leq 1\heartsuit\}$
          $S$
       ;  $d := d + 1$
          $\{d \leq 1?\}$
       $]$
$B : *[$  $\{1 \leq d\heartsuit\}$
          $T$
       ;  $d := d - 1$
          $\{1 \leq d?\}$
       $]$
$Inv : 0 \leq d \leq 2$
```

4.1 Symmetric Solution

It's easy to see that local correctness can be established through a guarded skip for each component. Again widening applies to prove non interference.

```
$Pre : d = 1$
$A : *[$  $\{d \leq 1\}$
          $S$
       ;  $d := d + 1$
       ;  if $d \leq 1 \rightarrow$ skip fi
          $\{d \leq 1\heartsuit\}$
       $]$
$B : *[$  $\{1 \leq d\}$
          $T$
       ;  $d := d - 1$
       ;  if $1 \leq d \rightarrow$ skip fi
          $\{1 \leq d\heartsuit\}$
       $]$
$Inv : 0 \leq d \leq 2$
```

From this point on we'll work, as in the previous example, on a *transformational approach* using guard strengthening lemma, to come up with a version where all synchronization is achieved through semaphores.

Finding a Q such that $Q \wedge 0 < s \Rightarrow d \leq 1$ can be done by simple predicate calculus.

$$Q \Rightarrow 0 < s \Rightarrow d \leq 1$$

$$
\begin{aligned}
&\equiv \{ \text{ algebra } \}\\
&\quad Q \Rightarrow 0 < s \Rightarrow 0 < 2 - d\\
&\Leftarrow \{ \text{ transitivity } \}\\
&\quad Q \Rightarrow s \leq 2 - d
\end{aligned}
$$

So we choose Q as weak as possible and annotate the component A with the new proof obligation.

```
Pre : $d = 1$
-----------------------------------------------------------------
$A : *[$  $\{d \leq 1\}$               | $B : *[$  $\{1 \leq d\}$
          $S$                          |           $T$
       ;  $d := d + 1$                 |        ;  $d := d - 1$
          $\{s \leq 2 - d?\}$          |        ;  **if** $1 \leq d \rightarrow$ **skip fi**
       ;  **if** $d \leq 1 \rightarrow$ **skip fi**  |  $\{1 \leq d\}$
          $\{d \leq 1\}$               |        $]$
       $]$                             |
-----------------------------------------------------------------
Inv : $0 \leq d \leq 2$
```

Since local correctness is not automatically established we calculate

$$wlp.(d := d + 1).(s \leq 2 - d) \;\equiv\; s \leq 1 - d$$

and decide to make it loop invariant for local correctness. Global correctness once again follows by widening.

The loop invariant is implied by a stronger Pre, namely one that makes $s \leq 0$ valid. Given the semaphore invariant $0 \leq s$ that is going to hold, we choose to conjunct Pre with $s = 0$. In order to have the loop invariant at the end of it we need to add assignment $s := s - 1$ somewhere between the $d := d + 1$ and the guarded skip. Again because our transformation is going in the semaphores direction, the election is simple. For the component B the situation is slightly less complicated, we give the correct annotated program and leave the details for the reader.

```
Pre : $d = 1 \wedge s = 0 \wedge t = 0$
-----------------------------------------------------------------------------
$A : *[$  $\{d \leq 1\}\ \{s \leq 1 - d\}$        | $B : *[$  $\{1 \leq d\}\ \{t < d\}$
          $S$                                    |           $T$
       ;  $d := d + 1$                           |        ;  $d := d - 1$
          $\{s \leq 2 - d\}$                     |        ;  $\{t \leq d\}$
       ;  **if** $d \leq 1 \rightarrow s := s - 1$ **fi** | ;  **if** $1 \leq d \rightarrow t := t - 1$ **fi**
          $\{d \leq 1\}\ \{s \leq 1 - d\}$        |           $\{1 \leq d\}\ \{t < d\}$
       $]$                                       |        $]$
-----------------------------------------------------------------------------
Inv : $0 \leq d \leq 2$
```

As in the previous example, progress requirement suggests some increments of s and t. The right place to do it is together with assignments to d. We get the next version applying guard strengthening lemma and adding these new statements.

$Pre : d = 1 \wedge s = 0 \wedge t = 0$	
$A : *[\ \ \{d \leq 1\}\ \{s \leq 1 - d\}$	$B : *[\ \ \{1 \leq d\}\ \{t < d\}$
S	T
$;\ \ d, t := d + 1, t + 1$	$;\ \ d, s := d - 1, s + 1$
$\{s \leq 2 - d\}$	$;\ \ \{t \leq d\}$
$;\ \ \textbf{if}\ 0 < s \rightarrow s := s - 1\ \textbf{fi}$	$;\ \ \textbf{if}\ 0 < t \rightarrow t := t - 1\ \textbf{fi}$
$\{d \leq 1\}\ \{s \leq 1 - d\}$	$\{1 \leq d\}\ \{t < d\}$
$]$	$]$
$Inv : 0 \leq d \leq 2$	

Now d is an auxiliary variable and hence it can be eliminated. Finally we can move to the P, V synchronization. The final version erasing all annotations is the following.

$Pre : s = 0 \wedge t = 0$	
$A : *[\ \ S$	$B : *[\ \ T$
$;\ \ V(t)$	$;\ \ V(s)$
$;\ \ P(s)$	$;\ \ P(t)$
$]$	$]$

Individual progress is guaranteed by the same arguments we used in the former example.

The final algorithm is the same obtained in [2]. However, our starting point is very different since the whole process was formally developed starting from a specification that was not biased toward a semaphore solution. In [2] the starting point is just a coordinate transformation of a semaphore.

4.2 Asymmetric Solution

We can consider the asymmetric solution [7] giving a less well-known program.

$Pre : d = 1$	
$A : *[\ \ \{d \leq 1\}$	$B : *[\ \ \{1 \leq d\}$
S	T
$;\ \ \textbf{if}\ d \leq 0 \rightarrow \textbf{skip}\ \textbf{fi}$	$;\ \ d := d - 1$
$\{d \leq 0\}$	$;\ \ \textbf{if}\ 1 \leq d \rightarrow \textbf{skip}\ \textbf{fi}$
$;\ \ d := d + 1$	$\{1 \leq d\}$
$\{d \leq 1\}$	$]$
$]$	
$Inv : 0 \leq d \leq 2$	

It's straightforward to obtain the following semaphore solution for this version applying exactly the same techniques we developed in former examples. The details are left to the reader.

$Pre : s = 0 \wedge t = 0$	
$A : *[\ \ S$	$B : *[\ \ T$
$;\ \ P(s)$	$;\ \ V(s)$
$;\ \ V(t)$	$;\ \ P(t)$
$]$	$]$

This example shows that a solution obtained using methodological development of multiprograms, can be brought into the realm of semaphores with very little cost. Since this asymmetric solution using conditional critical regions is new, our transformation also produces a new semaphore based solution.

5 Conclusions

We show through these examples that it is feasible to successfully use the results in formal development of multiprograms to obtain semaphore-based parallel programs. We found the method simple and fairly general. All the examples considered could be solved with the guard strengthening lemma only. The economy of tools involved in the transformations, together with the clear separation of concerns suggests a wide applicability and scalability of the method.

Although all the examples shown were only synchronized using guarded skips, the same technique could be applied to any kind of conditional atomic action. As a class exercise, we could easily solve a conditional region example involving non empty bodies in the synchronization.

The method presented in [10] uses a stronger version of the guard strengthening lemma which was not needed here. If for some examples the guard strengthening lemma is not powerful enough, it is straightforward to use this stronger version within our methodology.

Although the first two examples do appear in the literature, the construction of the programs is very different in particular the starting point of our transformation are programs which are formally developed. Given the fact that formal development techniques are useful to obtain different solutions of concurrent problems, our methodology can bring these new solutions into the semaphore synchronized programs in a fairly simple way. A very small example of this is the asymmetric solution to the phase synchronization problem.

References

1. G. R. Andrews, *A method for solving synchronization problems*, Science of Computer Programming 13, 4 (1-21) 1989.
2. G. R. Andrews, *Concurrent Programming: principles and practice*, Benjamin Cummings, 1991.
3. K. R. Apt and E-R. Olderog, *Verification of Sequential and Concurrent Programs*, Springer, 1991.
4. D. S. Buhăceanu and W. H. J. Feijen, *Formal derivation of an algorithm for distributed phase synchronization*, Information Processing Letters, 60:207-213, 1996.
5. E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, N. J., 1976.
6. E. W. Dijkstra, *A personal summary of the Owicki-Gries theory*, in: Selected Writings on Computing: A Personal Perspective, Springer, New York, 1982.
7. W. H. J. Feijen and A. J. M. van Gasteren, *On a Method of Multiprogramming*, Monographs in Computer Science, Springer, 1999.

8. S. S. Owicki and D. Gries, *An axiomatic proof technique for parallel programs*, Acta Informatica 6, 4 (319-340), 1976.
9. S. R. Faulk, D. L. Parnas, *On Synchronization in Hard-Real-Time Systems*, CACM 31(3): 274-287, 1988.
10. F. Schneider, *On Concurrent Programming*, Graduate Texts in Computer Science, Springer, 1997.