Programación Concurrente
Práctico 6: Derivación y Transformación

J. Blanco, N. Wolovick

1. Esta es la versión 3 de la derivación del algoritmo *Concurrent Vector Writing*.

Pre: $i = 0 \wedge j = 0$	
A: **do** $i \neq N \rightarrow$ $\quad x.i := 0;$ $\quad \{x.i = 0?\}$ $\quad i := i + 1$ **od** $\{i = N\}$	B: **do** $j \neq N \rightarrow$ $\quad \{i \leq j?N_0\}$ $\quad x.j := 1;$ $\quad j := j + 1$ **od** $\{j = N\}$
Inv: $\langle \forall k : 0 \leq k < i : x.k = 0 \rangle$	
Post: $\langle \forall k : 0 \leq k < N : x.k = 0 \rangle$	

   a) Muestre que si continuamos la derivación fortaleciendo ligeramente la aserción de la Nota 0 a $\{i < j\}$, el multiprograma resultante tiene riesgo de deadlock

   b) Realice una derivación alternativa, asegurando la Nota 0 con un skip guardado en B. Demuestre que el resultado obtenido no sufre de deadlock.

2. Considere un multiprograma con 23 componentes de tipo A y 47 de tipo B.

Pre: $s = 0$	
A: $*[s := s + 1]$	B: $*[s := s - 1]$

   Sincronice las componentes usando sólo skips guardados para que $0 \leq s \leq 100$ sea un invariante del sistema[1].

3. Sincronice las 2 componentes del siguiente multiprograma de manera tal que la postcondición implique $x = 0 \wedge y = 1$.

Pre: *true*	
A: $x := 0$ $;y := 0$	B: $x := 1$ $;y := 1$

   Ahora ataque el problema para $N = 3$, es decir, agregue sincronización al multiprograma de manera tal que la postcondición implique $x = 0 \wedge y = 1 \wedge z = 2$.

[1]Hint: vea el ejercicio 4 del práctico 5.

Pre: $true$		
A: $x := 0$ $;y := 0$ $;z := 0$	B: $x := 1$ $;y := 1$ $;z := 1$	C: $x := 2$ $;y := 2$ $;z := 2$

4. Reducir el grado de atomicidad para el siguiente multiprograma

Pre: $x \wedge y$	
A: $*$ [**if** $x \rightarrow$ **skip fi** $;\ x, y := \neg y, true$]	B: $*$ [**if** $y \rightarrow$ **skip fi** $;\ y, x := \neg x, true$]

   a) Demostrar que la solución no sufre de riesgo de deadlock total.

   b) ¿Podemos asegurar progreso individual[2]?

5. La siguiente es la *computation proper* de una barrera de N procesos, donde $e.i$ y $x.i$ son variables auxiliares que marcan respectivamente la entrada y salida de la barrera.

Pre: $\langle \forall i : 0 \leq i < N : \neg e.i \wedge \neg x.i \rangle$
Comp.k: $\vdots$ $e.k := true;$ $barrera.k$ $x.k := true;$ $\vdots$
Inv: $\langle \forall i : 0 \leq i < N : x.i \Rightarrow \langle \forall j : 0 \leq j < N : e.j \rangle \rangle$

   a) Derive la barrera del contador.

   b) Derive la barrera del contador implementada con N bits. Genere primero una versión *coarse grained*, para luego aplicar lemas de reducción del grado de atomicidad y obtener una versión *one-point*.

   c) Compare en términos prácticos las dos implementaciones.

6. A partir de la siguiente *computation proper*, crear sentencias que establezcan la Post. El resultado deberá ser simétrico y *one-point*. Notar que esta es la especificación de un problema de **elección de líder**.

Pre: $true$	
$C.i$: $y.i := B.i$	$C.j$: $y.j := B.j$
Post: $N.y.i + N.y.j = 1$?	

[2] Hint: pensar en las anotaciones –ocultas– del multiprograma.