EXAMEN DE VALIDACIÓN Y VERIFICACIÓN DEL SOFTWARE. 5/7/2021.

APELLIDOS Y NOMBRE: .....

1) **(70 pts)** Un cronómetro cuenta con un botón (b) de inicio/parada. El cronómetro puede estar parado p o en marcha $\neg p$. Además, el tiempo puede estar a cero 00:00:00 (c) o no. Formula los siguientes enunciados en LTL

   – Cuando pulsamos el botón, si estaba parado, se pone en marcha y viceversa, a no ser que lo mantengamos pulsado justo después.

   $$\Box(b \wedge \neg \bigcirc b \rightarrow (p \leftrightarrow \bigcirc \neg p))$$

   – Si mantenemos pulsado el botón dos situaciones seguidas, se para y se pone a cero inmediatamente después

   $$\Box(b \wedge \bigcirc b \rightarrow \bigcirc \bigcirc (c \wedge p))$$

   – Si no se pulsa el botón, el reloj permanece en el mismo estado (parado/en marcha) de un momento al siguiente

   $$\Box(\neg b \rightarrow (p \leftrightarrow \bigcirc p))$$

   – El reloj pasa a cero un número infinito de veces

   $$\Box\Diamond(\neg c \rightarrow \bigcirc c)$$

   – Nunca pulsamos el botón indefinidamente

   $$\neg\Diamond\Box b$$

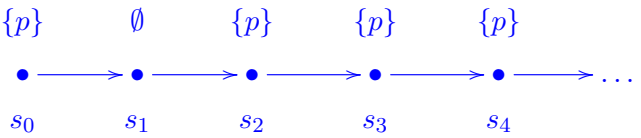
2) **(50 pts)**Dadas las fórmulas:

   $$\alpha \stackrel{def}{=} \neg p \,\mathcal{U}\, \Box p \qquad\qquad \beta \stackrel{def}{=} \Diamond\Box p$$

   demostrar cada dirección de la equivalencia o, si no se cumple, presentar un contraejemplo
   $\models \alpha \rightarrow \beta$ ¿se cumple? [X]-Sí [ ]-No
   Explicación: En general $\gamma \cup \delta$ implica $\Diamond\delta$ ya que el *until* conlleva que la condición de parada δ se alcance en algún punto. Como la condición de parada aquí es $\Box p$ tenemos que, si se cumple el *until*, entonces $\Diamond\Box p$.

$\models \beta \rightarrow \alpha$ ¿se cumple? [ ]-Sí [X]-No

Explicación: Como contraejemplo, podemos tomar una traza que cumple p en la situación inicial, inmediatamente $\neg p$ y a partir de ahí p siempre cierto:



Esta traza satisface β porque hay un punto $i = 2$ a partir del cual $\Box p$ es cierto. Sin embargo, no cumple α porque ningún punto que cumpla $\Box p$ tiene $\neg p$ en todos los puntos anteriores.

3) **(20 pts)** Explica brevemente en qué consiste la técnica de reducción de orden parcial (partial order reduction). Cuando la ejecución se enfrenta a varias opciones de forma no determinista (por ejemplo, varios procesos listos) y el orden entre esas opciones no es relevante, la reducción de orden parcial elige un orden lineal arbitrario prefijado. De lo contrario, si tenemos n alternativas, tendría que explorar $n!$ posibles órdenes de ejecución y, al no importar el orden, producirían todos el mismo resultado. Un caso típico en el que el orden de ejecución no es relevante es, por ejemplo, cuando varios procesos van a realizar operaciones sobre sus variables locales, no compartidas.

Satisfaction of a temporal formula
Let $M = s_0, s_1, \ldots$ with $i \geq 0$. We say that $M, i \models \alpha$ when:

- $M, i \models p$ if $p \in s_i$ (for $p \in \Sigma$)
- $M, i \models \Box\alpha$ if $M, j \models \alpha$ for all $j \geq i$
- $M, i \models \Diamond\alpha$ if $M, j \models \alpha$ for some $j \geq i$
- $M, i \models \bigcirc\alpha$ if $M, i+1 \models \alpha$
- $M, i \models \alpha \,\mathcal{U}\, \beta$ if there exists $n \geq i$, $M, n \models \beta$ and $M, j \models \alpha$ for all $i \leq j < n$.
- $M, i \models \alpha \,\mathcal{W}\, \beta$ if $M, i \models \Box\alpha$ or $M, i \models \alpha \,\mathcal{U}\, \beta$

Kamp's translation
Temporal formula α at time point i becomes $MFO(<)$ formula $\alpha(i)$

$$
\begin{aligned}
(p)(i) &\stackrel{def}{=} p(i) \\
(\neg\alpha)(i) &\stackrel{def}{=} \neg\alpha(i) \\
(\alpha \vee \beta)(i) &\stackrel{def}{=} \alpha(i) \vee \beta(i) \\
(\alpha \wedge \beta)(i) &\stackrel{def}{=} \alpha(i) \wedge \beta(i) \\
(\bigcirc\alpha)(i) &\stackrel{def}{=} \alpha(i+1) \\
(\Diamond\alpha)(i) &\stackrel{def}{=} \exists j \geq i : \alpha(j) \\
(\Box\alpha)(i) &\stackrel{def}{=} \forall j \geq i : \alpha(j) \\
(\alpha \,\mathcal{U}\, \beta)(i) &\stackrel{def}{=} \exists j \geq i : (\beta(j) \wedge (\forall k \in i..j-1 : \alpha(k))) \\
(\alpha \,\mathcal{V}\, \beta)(i) &\stackrel{def}{=} \forall j \geq i : (\beta(j) \vee (\exists k \in i..j-1 : \alpha(k)))
\end{aligned}
$$

- **(2 puntos del total de la asignatura)**: (equivale a práctica model checking) El siguiente código PROMELA:

```
/* Fichero ejemplo.pml */
bool wantP=false, wantQ=false;

active proctype P() {
  do
  :: printf("Non-critical section P\n");
     wantP=true;
     !wantQ;
 cs: printf("Critical section P\n");
     wantP=false;
  od
}

active proctype Q() {
  do
  :: printf("Non-critical section Q\n");
     wantQ=true;
     !wantP;
cs:  printf("Critical section Q\n");
     wantQ=false;
  od
}
```

pretende proteger con exclusión mutua el acceso a la sección crítica de dos procesos concurrentes `P, Q`.

1. Completa el siguiente comando para chequear la exclusión mutua en SPIN escribiendo la fórmula correspondiente en el hueco subrayado:

   ```
   spin -a -f '<>(P@cs && Q@cs)' ejemplo.pml
   ```

2. Tras ejecutar (correctamente) el comando de arriba se comprueba que no hay error de exclusión mutua, pero sí de interbloqueo. Explica brevemente cómo puede darse dicha situación con un ejemplo de ejecución entrelazada que lo provoque.

   Por ejemplo, el proceso `P` ejecuta el `printf`y después `wantP=true`, luego `Q` ejecuta su `printf` y `wantQ=true` y, al llegar a ese punto, las dos variables tienen valor `true` mientras que ambos procesos están interbloqueados esperando a que cada una de ellas se vuelva `false`.

   ¿Cómo arreglarías este problema sin perder la exclusión mutua? Comprobando primero la otra variable, antes de cambiar la propia y haciendo ambas operaciones atómicas:

```
active proctype P() {
  do
  :: printf("Non-critical section P\n");
     atomic{
       !wantQ;
       wantP=true;
     }
 cs: printf("Critical section P\n");
     wantP=false;
  od
}
```

y de forma análoga para el proceso `Q`.