EXAMEN DE VALIDACIÓN Y VERIFICACIÓN DEL SOFTWARE. 8/7/2015.

APELLIDOS Y NOMBRE: .....

1) **(30 %)** Para operar con una nueva tarjeta electrónica, el banco requiere que solicitemos su activación (evento a) telefónicamente o por internet. Una vez solicitada la activación el banco nos comunicará que la tarjeta pasa a estar operativa (el predicado p indica que está operativa). **Formula en LTL la siguiente condición**:

   > la tarjeta se activa *como mucho*[1] una única vez, nunca está operativa antes de activarse y siempre lo estará a partir de un momento no anterior a la activación.

   Una forma de expresar que la tarjeta se activa como mucho una vez podría ser:

   $$\neg a \ \mathcal{W} \ (a \wedge \bigcirc\Box\neg a)$$

   es decir, tenemos $\neg a$ hasta un punto en que se activa a y a partir del instante siguiente a ese punto $\neg a$ se vuelve cierto para siempre. Como hemos usado el "weak until" $\mathcal{W}$, también permitimos que $\neg a$ sea siempre cierto (la tarjeta no se activa). Modificar esta fórmula para garantizar que la tarjeta no esté operativa antes de activarse es muy sencillo. Basta requerir $\neg p$ en la condición de la izquierda del $\mathcal{W}$:

   $$(\neg a \wedge \neg p) \ \mathcal{W} \ (a \wedge \bigcirc\Box\neg a)$$

   Por último, requerir que la tarjeta se vuelve operativa una vez activada, aunque quizá con cierto retraso, se puede expresar añadiendo la condición $\Diamond\Box p$ (en algún punto, p se vuelve cierto para siempre) a la parte derecha del $\mathcal{W}$, obteniendo de este modo la respuesta final:

   $$\boxed{(\neg a \wedge \neg p) \ \mathcal{W} \ (a \wedge \bigcirc\Box\neg a \wedge \Diamond\Box p)}$$

2) **(30 %)** Demostrar que las fórmulas:

   $$\alpha \stackrel{def}{=} p \ \mathcal{U} \ \Diamond q \qquad\qquad \beta \stackrel{def}{=} \Diamond q$$

   son equivalentes o, si no lo son, encontrar un contraejemplo.

   Son equivalentes. Para comprobarlo, probaremos ambas direcciones.

   $\alpha \rightarrow \beta$

   Supongamos que $M, 0 \models p \ \mathcal{U} \ \Diamond q$. Entonces $\exists i \geq 0$ tal que $M, i \models \Diamond q$ y para todo $j = 0, \ldots, i-1$ tenemos $M, j \models p$. Como $M, i \models \Diamond q$ esto implica que $\exists k \geq i$ tal que $M, k \models q$. Como $k \geq i \geq 0$ hemos encontrado una posición $k \geq 0$ para la que $M, k \models q$ y por tanto $M, 0 \models \Diamond q$.

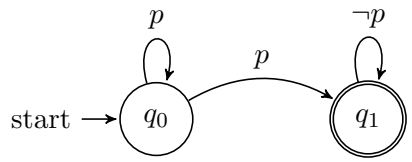
   $\beta \rightarrow \alpha$

   Es inmediato dado que ψ implica $\varphi \ \mathcal{U} \ \psi$, es decir, si la condición ψ de parada del "until" se cumple en el estado actual, entonces el "until" es trivialmente cierto. Por tanto, $\Diamond q$ implica directamente $p \ \mathcal{U} \ \Diamond q$.

[1] Es decir, el usuario podría no llegar a activar nunca la tarjeta.

4) **(40 %)** Los modelos capturados por el autómata de Büchi de abajo son expresables con una fórmula en LTL.

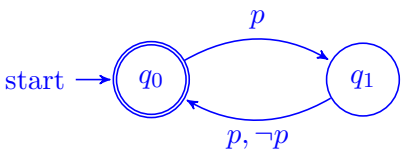
(a) Especifica dicha fórmula (o una equivalente).



fórmula LTL $= p \,\mathcal{U}\, (p \wedge \bigcirc\Box\neg p)$

(b) En general ¿es siempre posible encontrar una fórmula LTL que corresponda a cualquier autómata de Büchi? (justifica la respuesta).

No. Hay problemas representables con un autómata de Büchi que no se pueden expresar en LTL. Un ejemplo visto en clase es el conjunto de modelos en los que p es cierto en los estados pares pero varía libremente en los estados impares. Este conjunto de modelos se puede capturar con el autómata:



pero se ha demostrado que no es posible representarlo en LTL.

- **(1,3 puntos)** (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`.