EXAMEN DE VALIDACIÓN Y VERIFICACIÓN DEL SOFTWARE. 5/6/2019.

APELLIDOS Y NOMBRE: .....

1) **(50 pts)** Un alumno puede matricularse (m) en una asignatura, presentarse a examen (e) y aprobarla (a). Formula los siguientes enunciados en LTL

   – Para presentarse a examen, hay que estar matriculado

   $$\Box(e \rightarrow m)$$

   – Si una asignatura se aprueba, queda aprobada para siempre La solución más inmediata sería:

   $$\Box(a \rightarrow \Box a)$$

   Otra alternativa interesante es:

   $$\neg a \ \mathcal{W} \ \Box a$$

   – Si una asignatura se aprueba, ya no se vuelve uno a matricular de ella
   Si entendemos que en el momento mismo de aprobar ya se deja de estar matriculado, escribiríamos:

   $$\Box(a \rightarrow \Box\neg m)$$

   o quizá

   $$\Box(a \rightarrow \bigcirc\Box\neg m)$$

   si ocurre justo después. No es exactamente lo que dice la frase, pero podría admitirse como válido. Sin embargo, una respuesta más precisa sería interpretar "volver a matricularse" como una transición de $\neg m$ a m. De ese modo tendríamos que, tras aprobar, no volvemos a tener $\neg m \wedge \bigcirc m$. Esto es:

   $$\Box(a \rightarrow \neg\Diamond(\neg m \wedge \bigcirc m))$$

   – Existe un límite máximo de matrículas
   Podemos verlo como que m no puede darse infinitas veces

   $$\neg\Box\Diamond m$$

   o, lo que es equivalente, que a partir de un punto, dejamos de estar matriculados para siempre

   $$\Diamond\Box\neg m$$

– Para aprobar hay que presentarse a examen previamente

La dificultad de este enunciado reside en que, en LTL, no podemos expresar "previamente" sino sólo representar modalidades que hablan del futuro. Para ello, es más sencillo expresar el problema como una prohibición: no puede ser que termines aprobando sin que hasta entonces te hayas presentado a examen.

$$\neg(\neg e \; \mathcal{U} \; a)$$

Otra forma (equivalente) de verlo es usar el operador "release"

$$(e \; \mathcal{V} \; \neg a)$$

que se cumple si para cada estado que no cumple $\neg a$ hay un estado previo que cumple e. Esto también es equivalente a:

$$(\neg a \; \mathcal{W} \; (e \wedge \neg a))$$

2) **(40 pts)** Dadas la fórmulas

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

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, $\varphi \; \mathcal{U} \; \psi$ implica $\Diamond \psi$. Esto lo probamos en el Ejercicio 2 de las transparencias. Por tanto, $p \; \mathcal{U} \; p$ también implica $\Diamond p$.

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

Como contraejemplo, tomemos M con $S_0 = \emptyset$, $S_1 = \{p\}$ y el resto $S_j = \emptyset$ para $j > 1$. Está claro que $M, 0 \models \Diamond p$ porque hay un estado $j = 1$ para el que $M, j \models p$. Sin embargo, $M, 0 \not\models p \; \mathcal{U} \; p$ porque sólo hay un estado ($j = 1$) que cumple la condición de parada p, pero los estados anteriores ($j = 0$) no cumplen la parte izquierda del until, que es p también.

NOTA: en realidad se puede comprobar que $p \; \mathcal{U} \; p$ es equivalente a p.

3) **(10 pts)** Tenemos un programa que controla el acceso a un aparcamiento y sospechamos que en algunos casos puede no estar funcionando correctamente. Para verificar su funcionamiento, explica si usarías *model checking* o *theorem proving* y razona la respuesta.

En un caso en el que sospechamos que puede haber errores, lo más adecuado es utilizar *model checking*. Si usamos *theorem proving* y el programa no es correcto, no podremos encontrar una prueba de corrección y esto no nos da información, ya que un probador de teoremas puede no encontrar prueba en casos que sí son correctos. Por el contrario, si usamos un comprobador por modelos y existe error, con suerte (si el tamaño del autómata no explota exponencialmente) podremos obtener un contraejemplo, lo que además nos ayudaría a corregir el problema.