GRADO EN INGENIERÍA INFORMÁTICA
REPRESENTACIÓN DEL CONOCIMIENTO Y RAZONAMIENTO AUTOMÁTICO
Examen Parte I: Razonamiento Lógico. 8 de junio de 2026

Apellidos: _____ **Nombre:**_____

- **Puntuación**: este examen de la parte I supone un tercio del total de la teoría que, a su vez, es el 60 % de la asignatura. El máximo a obtener es, por tanto, $6/3 = \mathbf{2}$ **puntos** en el total de la asignatura. La puntuación de cada apartado del examen se mide en porcentaje de ese valor.
- Utilizar preferentemente el espacio reservado en el enunciado. En caso de equivocación, solicitar un formulario nuevo.

Ejercicio 1 (10 %). Dado el siguiente programa lógico P

$$\begin{array}{lcl} a & \leftarrow & not\ b \\ c & \leftarrow & a \\ b & \leftarrow & not\ c \end{array}$$

Indica cuáles son sus modelos clásicos mediante una tabla de verdad.

a	b	c	
0	0	0	
0	0	1	
0	1	0	×
0	1	1	×
1	0	0	
1	0	1	×
1	1	0	
1	1	1	×

Ejercicio 2 (30 %). Para cada uno I de los modelos clásicos obtenidos anteriormente, indica cuál es el reducto del programa anterior P^I, su modelo mínimo e indica si es o no un modelo estable. Usa tantas filas como necesites

modelo clásico I	programa reducto P^I	modelo mínimo de P^I	¿es estable? (sí/no)
$\{b\}$	$c \leftarrow a$ b	$\{b\}$	sí
$\{b, c\}$	$c \leftarrow a$	$\emptyset$	no
$\{a, c\}$	a $c \leftarrow a$	$\{a, c\}$	sí
$\{a, b, c\}$	$c \leftarrow a$	$\emptyset$	no

Ejercicio 3 (10 %).
Explica qué es la **ley de inercia** en Razonamiento sobre Acciones y por qué la Lógica Clásica no permite representarla adecuadamente.

La ley de inercia especifica que, en cada transición, un fluente mantiene el valor que tenía en el estado anterior si no existe evidencia de que deba cambiar. Es una regla por defecto que permite obtener una conclusión en ausencia de una evidencia (de cambio del fluente), de modo que si se añade nueva información, esa conclusión se puede retractar. Por tanto, requiere razonamiento no monótono, algo imposible en Lógica Clásica, cuya relación de inferencia es monótona.

Ejercicio 4 (40 %). Deseamos pagar la tarifa de un aparcamiento, que nos dan como un importe de `t>0` céntimos. Para ello, disponemos de varias monedas en el bolsillo, que indicamos con el predicado `bolsillo(N,X)` donde `N>0` es el número de monedas de cada tipo y `X` es el valor en céntimos de ese tipo de moneda. Por ejemplo, en el programa de abajo, la tarifa `t` son 235 euros y los hechos para `bolsillo` indican que tenemos 3 monedas de 50 cént., 4 de 20 cént., 2 de 10 cént. y otras 3 de 5 cént. Escribe un program en ASP que calcule todas las formas posibles de **pagar la tarifa exacta** `t` con las monedas disponibles, usando para ello el predicado `pagar(M,X)` que indica que, para pagar, usamos `M` monedas de valor `X` de las que tenemos en el bolsillo.

```
#const t=235.
bolsillo(3,50). bolsillo(4,20). bolsillo(2,10). bolsillo(3,5).
0 {pagar(1..N,X)} 1 :- bolsillo(N,X).
:- #sum{M*X,X:pagar(M,X)}!=t.

#show pagar/2.
```

Ejercicio 5 (10 %). Sobre el ejercicio anterior, deseamos ahora elegir aquellas soluciones que usen el mayor número total de monedas, para quitar calderilla del bolsillo. Escribe la(s) sentencia(s) ASP que necesites añadir al programa anterior para maximizar el número de monedas en el pago.

```
#maximize{M,X:pagar(M,X)}.
```