REPRESENTACIÓN DEL CONOCIMIENTO Y RAZONAMIENTO AUTOMÁTICO

Examen 24/5/2018. **Apellidos, nombre:** .....

Ejercicio 1. Dado el siguiente programa lógico proposicional P:

$$carne \leftarrow not\ patatas \quad (1)$$
$$pescado \leftarrow not\ carne \quad (2)$$
$$patatas \leftarrow pescado \quad (3)$$

1a) Indica cuáles son sus modelos clásicos mediante una tabla de verdad.

La primera regla corresponde a $\neg patatas \rightarrow carne$ que equivale a $patatas \vee carne$. Del mismo modo, las otras dos reglas se corresponden con las disyunciones $carne \vee pescado$ y $\neg pescado \vee patatas$, respectivamente.

patatas	*carne*	*pescado*	(1)	(2)	(3)	(1) ∧ (2) ∧ (3)
0	0	0	0	0	1	0
0	0	1	0	1	0	0
0	1	0	1	1	1	1
0	1	1	1	1	0	0
1	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Es decir, obtenemos cuatro modelos clásicos $I_0 = \{carne\}$, $I_1 = \{patatas, pescado\}$, $I_2 = \{patatas, carne\}$ e $I_3 = \{patatas, carne, pescado\}$.

1b) Para cada modelo clásico I obtenido anteriormente, obtén el reducto P^I e indica, a partir de él, si I es modelo estable (*stable model*).

modelo clásico I	reducto P^I	estable (sí/no)
$I_0 = \{carne\}$	$carne \leftarrow$ $patatas \leftarrow pescado$	Sí
$I_1 = \{patatas, pescado\}$	$pescado \leftarrow$ $patatas \leftarrow pescado$	Sí
$I_2 = \{patatas, carne\}$	$patatas \leftarrow pescado$	No
$I_3 = \{patatas, carne, pescado\}$	$patatas \leftarrow pescado$	No

Ejercicio 2. Un robot debe recorrer una habitación organizada en $m \times n$ baldosas, evitando los obstáculos y llegando a una posición meta determinada. El robot comienza siempre en la posición $0, 0$ y la posición `X,Y` de la meta viene dada por un hecho de entrada del estilo `meta(X,Y)`. Nos proporcionan el siguiente código ASP incompleto:

```
#include <incmode>.
#program base.
    fila(0..m-1). col(0..n-1). robot(0,0,0).
    pos(X,Y) :- fila(X), col(Y).
    adyacente(X,Y,X+1,Y) :- X<m-1.
    adyacente(X,Y,X-1,Y) :- X>0.
    adyacente(X,Y,X,Y+1) :- Y<n-1.
    adyacente(X,Y,X,Y-1) :- Y>0.

#program step(t).

     ____________________________________ :- ____________________  % Choice rule
     :- robot(X,Y,t), obstaculo(X,Y).
#program check(t).
     goal(t) :- ______________________________
     :- query(t), not goal(t).
#show robot/3.
```

2a) Las 4 reglas del predicado `adyacente` generan un error acerca de las variables `X` e `Y`. Explica cuál puede ser el problema y propón una solución sobre el código.
Explicación: Las variables `X` e `Y` no son seguras (*safe*) dado que no aparecen en ningún predicado en el cuerpo positivo. Haría falta añadir, por ejemplo, `pos(X,Y)` a todos los cuerpos de las reglas de adyacente:

```
adyacente(X,Y,X+1,Y) :- pos(X,Y), X<m-1.
adyacente(X,Y,X-1,Y) :- pos(X,Y), X>0.
adyacente(X,Y,X,Y+1) :- pos(X,Y), Y<n-1.
adyacente(X,Y,X,Y-1) :- pos(X,Y), Y>0.
```

2b) Añade una *choice rule* para hacer que el robot salte a alguna posición adyacente.

```
1 {robot(X,Y,t): adyacente(U,V,X,Y) } 1 :- robot(U,V,t-1).
```

2c) Completa el cuerpo para la regla del predicado `goal(t)`.

```
:- robot(X,Y,t), meta(X,Y).
```

2d) Alguien nos plantea evitar que el robot camine por sitios ya recorridos y queremos valorar que mejore en eficiencia sin perder soluciones válidas. ¿Valdría la pena esa opción?

- ☐ Sí. Explica por qué es interesante y cómo modificarías el código para lograrlo.
- ☐ No. Explica qué inconveniente(s) hace(n) desaconsejable esa opción.

Explicación: La respuesta puede ser positiva o negativa, dependiendo del razonamiento y el código planteado en la explicación. Evitar posiciones visitadas pierde soluciones válidas, pero no altera las soluciones mínimas. Esto es así porque un plan de longitud mínima nunca vuelve sobre el camino recorrido. La modificación será más o menos eficiente dependiendo de cómo se plantee. Por ejemplo, si evitamos volver por cualquier sitio anterior `T<t`

```
:- robot(X,Y,t), robot(X,Y,T), T<t.
```

aumentamos bastante el tamaño del programa ground lo que, dependiendo del tamaño o la forma de la habitación, puede ser peor que la ganancia obtenida evitando acciones innecesarias. Ahora bien, si sólo prohibimos volver a donde estuvimos hace 2 pasos

```
:- robot(X,Y,t), robot(X,Y,t-2), t>1.
```

el programa ground no aumenta en exceso, mientras que se evita volver por lo andado hasta cierto punto.