REPRESENTACIÓN DEL CONOCIMIENTO Y RAZONAMIENTO AUTOMÁTICO

Examen 23/5/2019. **Apellidos, nombre: ....**

1a) **(2 pts.)** Dado el siguiente programa lógico proposicional P:

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ q & \leftarrow & not\ p \\ r & \leftarrow & p, not\ r \end{array}$$

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

En lógica clásica, las dos primeras reglas son equivalentes a $p \vee q$ mientras que la tercera sería $p \wedge \neg r \rightarrow r$ que es equivalente a $\neg p \vee r$.

p	q	r	$p \vee q$	$\neg p \vee r$	*modelo*
0	0	0	0	1	
0	0	1	0	1	
0	1	0	1	1	×
0	1	1	1	1	×
1	0	0	1	0	
1	0	1	1	1	×
1	1	0	1	0	
1	1	1	1	1	×

Es decir, obtenemos cuatro modelos clásicos $\{q\}, \{q, r\}$, $\{p, r\}$, y $\{p, q, r\}$.

1b) **(3 pts.)** Para cada modelo clásico I obtenido anteriormente, obtén el programa reducto P^I correspondiente, su modelo mínimo y, finalmente, indica si I es modelo estable (*stable model*). Usa tantas filas como precises.

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

2) En un tablero de ajedrez de `n x n` con $n \geq 1$ se desean colocar $m > 0$ caballos sin que se ataquen entre sí, usando el predicado `horse(X,Y)` para indicar que la celda `X, Y` contiene un caballo.

2a) **(3 pts.)** Completa el siguiente código ASP para resolver el problema:

```
#const n=3.
#const m=5.
row(1..n). col(1..n).
cell(X,Y) :- row(X), col(Y).

% Generar posibles colocaciones de caballos (completar)

m { horse(X,Y) : cell(X,Y) } m.

% prohibir ataques entre ellos (completar)
:- horse(X,Y), horse(X+1,Y+2).            % (*)
:- horse(X,Y), horse(X-1,Y+2).
:- horse(X,Y), horse(X+2,Y+1).
:- horse(X,Y), horse(X-2,Y+1).
```

NOTA: en algunas soluciones, además de las cuatro restricciones de arriba, otras cuatro más fueron añadidas:

```
:- horse(X,Y), horse(X+1,Y-2).
:- horse(X,Y), horse(X-1,Y-2).
:- horse(X,Y), horse(X+2,Y-1).
:- horse(X,Y), horse(X-2,Y-1).
```

Esto es correcto, pero innecesario. Pongamos, por ejemplo, la primera de estas cuatro reglas adicionales:

```
:- horse(X,Y), horse(X+1,Y-2).
```

Esto es lo mismo que:

```
:- horse(X,Y), horse(X',Y'),X'=X+1,Y'=Y-2.
```

que es igual que:

```
:- horse(X,Y), horse(X',Y'), X'-1=X, Y'+2=Y.
```

es decir:

```
:- horse(X'-1,Y'+2), horse(X',Y').
```

que no deja de ser otra forma de escribir la regla que hemos marcado como `(*)`. Lo mismo sucede con las otras tres.

2b) **(1 pt.)** En la restricción marcada arriba como `(*)`, ¿ sería necesario comprobar que `X+1` e `Y+2` no se salgan del rango de 1 a `n`? Razona la respuesta.

No es necesario. Lo normal es que ya generemos hechos para el predicado `horse(X,Y)` a partir de casos de `cell(X,Y)`, que son posiciones válidas (como en la solución propuesta arriba). Pero incluso si eso no fuese así y pudiésemos generar `horse(X+1,Y+2)` fuera del tablero, lo lógico sería añadir otra restricción aparte para prohibirlo.

```
:- horse(X,Y), not cell(X,Y).
```

2b) **(1 pt.)** ¿Cuántos casos *ground* (esto es, sin variables) generará la regla `(*)` cuando `n=3`? Razona la respuesta.

En general, tendríamos $n^2 = 9$ posibles casillas o pares `X,Y`. Los valores de `X+1` e `Y+2` vienen fijados por `X,Y` y, por tanto, no generan más combinaciones. Ahora bien, como se explicó antes, también han de estar dentro del tablero, lo que obliga a que `X` $\leq n-1$ e `Y` $\leq n-2$, así que tendríamos sólo $(n-1) \times (n-2) = 2 \times 1 = 2$ posibles combinaciones. En nuestro ejemplo, serían

```
:- horse(1,1), horse(2,3).
:- horse(2,1), horse(3,3).
```

Movimientos del caballo de ajedrez

El caballo de ajedrez se desplaza (y por tanto, ataca) a las posiciones a las que se pueda acceder trazando una `L` de tres casillas. En la figura de abajo se muestra un caballo en la zona central de un tablero estándar de 8×8 y las posiciones a las que ataca se corresponden con las 8 casillas que contienen una cabeza de flecha.