**REPRESENTACIÓN DEL CONOCIMIENTO Y RAZONAMIENTO AUTOMÁTICO**
Examen 5/7/2019. **Apellidos, nombre:** .....

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

$$\begin{aligned} p &\leftarrow not\ q \\ q &\leftarrow not\ r \\ r &\leftarrow not\ p \end{aligned}$$

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

En lógica clásica, las tres reglas corresponderían respectivamente $p \vee q$, $q \vee r$ y $p \vee r$.

| $p$ | $q$ | $r$ | $p \vee q$ | $q \vee r$ | $p \vee r$ | *modelo* |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 1 | 1 | × |
| 1 | 0 | 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 1 | 1 | 1 | × |
| 1 | 1 | 0 | 1 | 1 | 1 | × |
| 1 | 1 | 1 | 1 | 1 | 1 | × |

Es decir, obtenemos cuatro modelos clásicos $\{q, r\}, \{p, r\}$, $\{p, q\}$, 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) |
|---|---|---|---|
| $\{q, r\}$ | $r \leftarrow$ | $\{r\}$ | no |
| $\{p, r\}$ | $p \leftarrow$ | $\{p\}$ | no |
| $\{p, q\}$ | $q \leftarrow$ | $\{q\}$ | no |
| $\{p, q, r\}$ | (vacío) | $\emptyset$ | no |

El programa no tiene modelos estables.

2) En la lista de invitados de una boda, la familia nos facilita una lista con hechos para el predicado $odia(X, Y)$ donde $X$ e $Y$ son personas numeradas de 1 a $n = m * c$ y $m$ es el número de mesas con capacidad para $c$ personas cada una. Queremos obtener asignaciones a mesas en las que la gente no coincida con ninguna persona a la que odie. Las soluciones deben expresarse en términos del predicado $sienta(X, M)$, que sienta a la persona $X$ en la mesa $M$.

2a) **(3 pts.)** Nos proporcionan el siguiente código ASP con algunos datos de ejemplo y la restricción principal. Completa el código para resolver correctamente el problema:

```
#const m=3.
#const c=3.
#const n=m*c.
persona(1..n).
mesa(1..m).
odia(3,5).
odia(1,9).
odia(8,9).
odia(2,4).
:- sienta(X,M), sienta(Y,M), odia(X,Y).        % (*)
1 { sienta(X,M):mesa(M) } 1 :- persona(X).
:- mesa(M), not c {sienta(X,M):persona(X)} c.
```

2b) **(1 pt.)** Explica al menos dos situaciones en los datos de entrada que provocarían que el problema no tenga solución.

Un ejemplo trivial que no tiene solución sería si una persona se odia a sí misma. Otros ejemplos serían si hay una persona odiada por todo el mundo, o si toda persona odia a alguien.

2c) **(1 pt.)** Dados los hechos de entrada y la restricción `(*)` que se proporciona en el apartado 2a) ¿Cuántos casos *ground* (esto es, sin variables) generará la regla `(*)`? Razona la respuesta.

En el programa de ejemplo, tenemos 4 hechos para el predicado $odia(X, Y)$ mientras que el número de mesas es $m = 3$. Así pues, cuatro pares de valores $(X, Y)$ por 3 posibles valores para $M$ nos da 12 casos ground.