Universidad de Granada
Departamento de Ciencias de la Computación
e Inteligencia Artificial

DECSAI

Curso 2015/16

Técnicas de los Sistemas Inteligentes

Relación de Problemas 2

Ejercicios básicos de Prolog.

1. **Relaciones familiares.** Crear una base de datos familiar añadiendo hechos, representando el árbol familiar con los siguiente predicados

   `5.a. male(X) : X is a male`

   `5.b. female(X): X is a female`

   `5.c. parent(X,Y) : X is a parent of Y`

   Implementar predicados representando las siguientes relaciones:

`mother(X,Y)`	`brother(X,Y)`
`father(X,Y)`	`sister(X,Y)`
`grandmother(X,Y)`	`siblings(X,Y)`
`grandfather(X,Y)`	`uncle(X,Y)`
`ancestor(X,Y)`	`cousing(X,Y)`

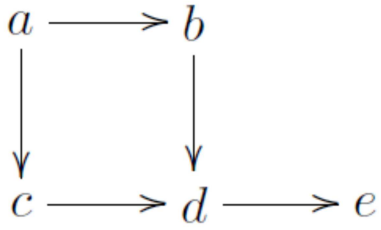
   Añadir hechos sobre tu familia y comprobar que los predicados funcionan correctamente.

2. **Unificación.** Intentar primero este ejercicio a mano sin la ayuda de ECLiPSe, verificar las soluciones después comprobándolas. Especificar si cada consulta tiene éxito o falla; y si tiene éxito, especificar qué se asigna a las variables en la unificación; si falla explicar por qué.

`?- X = hans.`	`?- 2 + 1 = 3.`
`?- anja = hans.`	`?- f(X,a) = f(a,X).`
`?- X = hans, X = Y.`	`?- likes(anja, X) = likes(X, hans).`
`?- X = happy(hans).`	`?- A = b(c).`
`?- X is Y.`	`?- a = b(c).`

3. **Representación para coloreado.** Representar la relación de adyacencia para un problema de coloreado de mapas. Dado el mapa de Australia (por ejemplo, aunque podría ser cualquier otro), crear una base de hechos en Prolog que represente las distintas regiones, los distintos colores posibles y la relación de adyacencia entre regiones.Comprobar la correcta representación haciendo consultas de la forma: `color(rojo), color(X), region(q), region(R), adyacente(q,wa), adyancente(X,wa), adyacente(X,Y).`

4. **Grafos dirigidos.** Un grafo dirigido puede representarse almacenando información sobre los arcos y nodos como hechos de PROLOG. Modelar el siguiente arco dirigido con sus respectivos hechos.



Implementar un predicado `path(X,Y)` que es cierto si existe un camino desde el nodo X hasta el nodo Y. Probar la ejecución de las siguientes consultas y trazar su ejecución

```
path(a,c). path(b,Y). path(f,g).
```

Para trazar la ejecución de un programa en ECLiPSe usar la herramienta **Tracer**, en *Tools>Tracer*. Aparecerá una nueva ventana que permite ejecutar paso a paso un programa. Probar básicamente los botones Creep y Skip. Ver la documentación de cómo depurar y trazar programas en ECLiPSe en http://eclipseclp.org/doc/tutorial/tutorial037.html.

5. **Búsqueda de caminos**.Implementar un programa PROLOG para ayudar a planear rutas. El programa usa la siguiente base de conocimiento, mostrando distancias entre principales ciudades de Alemania.

`Hamburg, Bremen, 80`	`Hamburg, Hannover,110`
`Hamburg, Berlin, 230`	`Bremen, Hannover,100`
`Bremen, Dortmund, 200`	`Hannover, Kassel,140`
`Hannover, Nuernberg, 380`	`Dortmund, Kassel,130`
`Dortmund, Koeln, 80`	`Kassel, Wuerzburg,180`
`Kassel, Frankfurt, 180`	`Frankfurt, Wuerzburg,110`
`Nuernberg, Muenchen, 160`	

`5.a.` Representar las distancias con la estructura Prolog `dist(city0,city1,km).` Comprobar la implementación con consultas como `?- dist(dortmund,bremen,Km), ?- dist(From,To,180), o ?- dist(From,To,Km).`

`5.b.` Definir el predicado route(From,To) que debe ser cierto si hay una ruta entre From y To. Comprobarlo con consultas como route(muenchen,berlin), route(ulm,halle) y finalmente, route(Frankfurt,To).

`5.c.` Definir un predicado ternario route(From,To,Route) que tenga la misma funcionalidad que route/2 con los dos primeros parámetros. Además, considerar que el tercer parámetro, Route, contiene las ciudades ya visitadas con el objeto de evitar ciclos (e.d. visitar una ciudad más de una vez). Para comprobar su funcionamiento usar consultas instanciadas y no instancias, p.e, `?- route(dortmund,X,[kassel]).`

`5.d.` Implementar el predicado Prolog shortestPath(From,To,Route,Km) que debe ser cierto si la distancia mínima entre las ciudades From y To es Km pasando por Route. La comprobación debería incluir, p.ej, shortestPath(kassel,bremen,[hannover],X).

6. **Gestión de tareas.**Considerar un conjunto de tareas para las que se ha definido la relación de precedencia. Por ejemplo:

Tarea	**Descripción**	**Duración**	**Predecesoras**
A	Levantar muros	7	Ninguna
B	Carpintería de tejado	3	A
C	Poner tejado	1	B
D	Instalación eléctrica	8	A
E	Pintado fachada	2	C,D
F	Ventanas	1	C,D
G	Jardín	1	C,D
H	Techado	3	A
I	Pintado interior	2	F,H

Implementar un programa PROLOG con las siguientes características:

1. Utilizar estructuras para representar la información de cada fila de la tabla.
2. A partir de la información de las estructuras definidas, crear cuatro listas: una para los identificadores de tarea, otra para las descripciones, otra para las duraciones y otra para las predecesoras (esta última es una lista de listas).
3. Escribir un predicado que guarde en una lista de listas las sucesoras de cada tarea, por ejemplo, `[[sucesoras_de_A],[sucesoras_de_B],…..].` .(Pista: puede usarse el predicado findall/3).
4. Escribir un predicado similar pero considerando ahora que el contenido de cada nodo de la lista es una tupla `succ(Tarea,Lista)` en la que el primer elemento es una tarea y el segundo es la lista de sus sucesoras, por ejemplo, `[succ(A,[sucesoras_de_A]),succ(B,[sucesoras_de_B],…..].`
5. Escribir un predicado que muestre en pantalla las sucesoras de cada tarea. Por ejemplo:

   `Sucesoras de Levantar Muros: <sucesoras>.`

   `Sucesoras de Carpintería de tejado: <sucesoras>`

   …
6. Modificar ese programa para que muestre en pantalla las sucesoras de una tarea con una duración mayor a una dada.