Práctica 3

Representación del Conocimiento

Práctica de Prolog

Autores: Álvaro López García,
Jairo González Gómez,
Nicolas Rodrigo Pérez.

Fecha: 7 de noviembre de 2021

Introducción.

En esta práctica vamos a hacer una introducción al lenguaje Prolog para demostrar que cualquier problema puede ser escrito como un problema de lógica.

Prolog es una abreviatura de "Programming in Logic" y su objetivo es ser un lenguaje para la computación simbólica lógica, es decir, esta diseñado para problemas que se pueden describir con datos y relaciones.

Para realizar la introducción necesaria al lenguaje, seguiremos el libro "Prolog Programming for Artificial Intelligence" de Ivan Bratko. Todos los ejemplos están tomados del libro, o han sido ligeramente modificados.

El siguiente código expresa las relaciones parentales de varias personas.

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

Como se vio en teoría, el lenguaje de primer orden consta de predicados, variables y funciones. En prolog solo se pueden definir funciones de aridad cero, esto es constantes. Estas se definen con cadenas de texto que no estén seguidas por paréntesis. Por ejemplo, en el código anterior "pam" es una constante (también llamadas "átomos").

Si una cadena de texto viene seguida de un paréntesis, entonces esta cadena es un predicado. Se hace notar que en la sintaxis Prolog no hace falta definir cuales son predicados ni su aridad, esta queda definida por su uso. En el ejemplo anterior, "parent" (progenitor) es un predicado de aridad dos.

Finalmente queda definir relaciones, que se expresarán como *cláusulas de Horn*, fórmulas del tipo:

$$P_1 \wedge \ldots \wedge P_n \longrightarrow P_{n+1}.$$

En el siguiente ejemplo, vamos a definir un predicado "mother" que tiene aridad 2 y que expresa que un progenitor que sea del sexo femenino es una madre.

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
sex( pam, feminine).
sex( tom, masculine).
sex( bob, masculine).
sex( pat, feminine).
mother(X,Y):- parent(X,Y), sex(X, feminine).
```

Toda cadena de texto que empiece por mayúscula es una variable, es decir, que puede sustituirse por cualquier constante y esa sustitución afecta a todas las apariciones de la variables.

Con esto tenemos *una base de conocimiento* y, con esta base de conocimiento podemos hacer consultas. Cada linea del siguiente código representa una consulta:

```
parent(X,Y).
parent(X,Y), parent(Y,Z).
parent(pam,Y), parent(Y,Z).
mother(X,Y).
mother(X,jim).
```

Como resumen: este ejemplo ilustra varias cosas:

- **Un programa de prolog consiste en diferentes reglas, cada una terminada en punto.**
- **Los argumentos de la relación pueden ser objetos concretos («átomos») o variables. Los primeros se escriben en minúscula y los segundos en mayúsculas.**
- **Las preguntas que se pueden plantear incluyen la conjunción y se expresa utilizando la coma.**

1) ***Escribir que devuelve cada una de las consultas anteriores. ¿Puedes poner alguna consulta que no devuelva ningún resultado? ¿Qué es lo que se devuelve en ese caso?***

Las consultas anteriores devuelven respectivamente:

```
?- parent(X,Y).
X = pam,
Y = bob ;
X = tom,
Y = bob ;
X = tom,
Y = liz ;
X = bob,
Y = ann ;
X = bob,
Y = pat ;
X = pat,
Y = jim.
```

```
?- parent(X,Y), parent(Y,Z).
X = pam,
Y = bob,
Z = ann ;
X = pam,
Y = bob,
Z = pat ;
X = tom,
Y = bob,
Z = ann ;
X = tom,
Y = bob,
Z = pat ;
X = bob,
Y = pat,
Z = jim ;
false.
```

```
?- parent(pam,Y), parent(Y,Z).
Y = bob,
Z = ann ;
Y = bob,
Z = pat.
```

```
?- mother(X,Y).
X = pam,
Y = bob ;
X = pat,
Y = jim.
```

```
?- mother(X,jim).
X = pat.
```

Si que es posible realizar consultas que no devuelven resultado, devolviendo false en dicho caso, como por ejemplo:

```
?- parent(X,pam).
false.
```

Cambiemos el siguiente código para la siguiente familia:

```
parent( pamela, leo).
parent( torn, leo).
parent( torn, isa).
parent( leo, ana).
parent( leo, patricia).
parent( patricia, jaime).
woman(pamela).
woman(isa).
woman(ana).
woman(patricia).
man(torn).
man(leo).
man(jaime).
sister(X,Y):- parent(Z,X), parent(Z,Y),woman(X).
```

2) ***Definir feliz si se tiene un hijo, comprobar si leo es feliz. Definir una regla que si una persona que tenga un hijo y ese hijo tenga una hermana entonces tiene pareja. Definir una regla para comprobar si alguien es ancestro de otra persona.***

```
%feliz/1
feliz(X):- parent(X,_).
%pareja/1
pareja(X):- parent(X,Y), sister(_,Y).
%ancestro/2
ancestro(X,Y):- parent(X,Z), parent(Z,Y).
```

Prolog también admite como constante números, tanto enteros como decimales y tiene estructuras como listas. Las listas se definen entre corchetes, y para utilizarlas en una regla se deben separar en dos partes.

```
es_miembro(Elemento, [Cabeza| Cola]):- Elemento = Cabeza.
es_miembro(Elemento, [Cabeza| Cola]):- es_miembro(Elemento, Cola).
```

Esta implementación coincide formalmente con la implementación de una lista enlazada en cualquiera de los lenguajes de programación imperativos (Java, Python, etc..).

3) ***Hacer un predicado de aridad tres, `posicion(Elemento, Lista, Posicion)` que devuelva verdadero si en la posicion Posicion de la lista Lista esta el Elemento.***

```
%posicion/3
posicion(E,L,P):- es_miembro(E,L,P).
%es_miembro/3
es_miembro(Elemento, [Cabeza| Cola], P):- Elemento = Cabeza, P is 0.
es_miembro(Elemento, [Cabeza| Cola], P):- P1 is P - 1, es_miembro(Elemento, Cola, P1).
```

Simplemente mencionar que estas funciones están implementadas en Prolog y se llaman *member* y *nth0*.

Para acabar, en las reglas se puede utilizar el operador `$;$` para denotar la operacion `OR`.

El problema de los sobrinos de Donald y Daisy.

Donald y Daisy cogieron a sus sobrinos de edades 4,5 y 6 para dar una vuelta. Cada sobrino llega una camiseta diferente, que tiene un dibujo y un color distinto. Se da la siguiente información:

- **Huey tiene menos edad que el sobrino con la camiseta verde,**
- **El sobrino de 5 años tiene la camiseta con el dibujo de un camello, Dewey lleva una camiseta amarilla,**
- **Louie lleva una camiseta con una jirafa,**
- **La camiseta con el dibujo de un oso panda no es blanca.**

Se ha dado el siguiente código prolog incompleto:

```
ninos(S):-
%nombre, edad, animal, color
member([dewey,_,_ ,amarillo], S),
member([_,5,camello,_],S),
member([louie,_,jirafa, _], S),
todos([dewey, louie, huey], S,0),
todos([4, 5, 6], S, 1),
todos([oso, jirafa, camello], S, 2),
todos([verde, amarillo, blanco], S, 3),
restriccion(huey, verde, S),
(member([_,_,oso,amarillo], S);member([_,_,oso,verde], S)).

% Huey tiene menos edad que el sobrino con la camiseta verde.
restriccion(Nombre, Color, S):-
(member([Nombre,4, _,_], S), member([_, 5, _,Color],S));
(member([Nombre,4, _,_], S), member([_, 6, _,Color],S));
(member([Nombre,5, _,_], S), member([_, 6, _,Color],S)).

todos([X], [L| _], Pos):- nth0(Pos, L, X).
todos([X], [ _ |L2], Pos):- todos([X], L2, Pos).
todos([X|Y], [L], Pos):- nth0(Pos, L, X), todos(Y, [L], Pos).
todos([X|Y], [L1|L2], Pos):- nth0(Pos, L1, X), todos(Y,L2, Pos).
todos([X|Y], [L1|L2], Pos):- todos([X], L2, Pos), todos(Y,[L1|L2], Pos).
```

4) ***Completar el código de forma que la consulta `ninos([X,Y,Z])` devuelva la solución.***

Se ha añadido al final del predicado `ninos()` el operador `! (cut)` para evitar que la consulta genere nuevas combinaciones de la solución, siendo ahora tal que:

```
ninos(S):-
%nombre, edad, animal, color
member([dewey,_,_ ,amarillo], S),
member([_,5,camello,_],S),
member([louie,_,jirafa, _], S),
todos([dewey, louie, huey], S,0),
todos([4, 5, 6], S, 1),
todos([oso, jirafa, camello], S, 2),
todos([verde, amarillo, blanco], S, 3),
restriccion(huey, verde, S),
(member([_,_,oso,amarillo], S);member([_,_,oso,verde], S)),!.
```

El problema del robot de limpieza.

Para programar el comportamiento de un robot ROMBLA™(a partir de ahora se llamara el robot), se ha decidido modelar una habitación como una cuadrícula de

tres por tres. Dentro de la habitación, hay un montón de basura y la papelera. El robot puede realizar tres acciones:

- **ir de una posición de la habitación a otra;**
- **recoger basura, si el robot está en las mismas coordenadas que la cuadrícula;**
- **soltar basura si esta en las mismas coordenadas que la papelera.**

Se plantea programar en prolog el robot y la empresa ExpertLogic da el siguiente código:

```
1  % El estado se compone de tres posiciones, la posicion del robot, de
2  % la papelera y la basura.
3  estado(cuadricula(1,1),cuadricula(1,2), cuadricula(1,3)).
4
5  % El robot puede moverse entre las distintas posiciones.
6  accion(estado(Pos1,Pos2, Pos3), ir(Pos1,Pos4), estado(Pos4, Pos2, Pos3)).
7  % El robot puede recoger la basura si esta encima de ella.
8  accion(estado(Pos1,Pos2, Pos3), recoger, estado(Pos1, Pos2, cargada)).
9  % El robot puede soltar la basura si esta cargada y estamos al lado de la papelera.
10 accion(estado(Pos1,Pos1, cargada), soltar, estado(Pos1, Pos1, en_papelera)).
11
12 %% Los planes empiezan en un estado y acaban en otro.
13 plan(Estado, Estado, []). plan(Inicio, Fin, [Accion1|Resto]) :-
14 accion(Inicio, Accion1, Estado), plan(Estado,Fin,Resto).
15
16 %%:-plan(estado(cuadricula(1,1), cuadricula(1,2),cuadricula(1,3)),
   ↪  estado(_,_,en_papelera), Plan)
17 %%,write(Plan).
```

5) *Se da el siguiente código prolog, ¿por que no funciona? Corregir el código y comentar las razones por las que no funcionaba (pista: utilizar la función "trace").*

No funciona (en mayor medida) porque los tres predicados `accion/3` no están colocados empezando por el mas concreto y terminando por el mas genérico. Cuando realizamos una consulta en prolog, se evaluan "de arriba a abajo" y de forma secuencial los predicados de nuestro script, de modo la consulta tendrá en cuenta el primer predicado que se encuentre y que satisfaga las condiciones que buscamos.

Es por esto, que si colocamos el caso mas genérico al principio (en nuestro caso, aquel en el que el robot simplemente se mueve de una casilla a otra), nunca llegarán a tenerse en cuenta aquellos casos más concretos (nuevamente en este caso, aquellos que tienen constantes en la cabecera).

Por lo que podemos concluir que, conforme a este programa, nuestro robot estará siempre moviéndose entre casillas arbitrariamente, ya que cualquier estado satisface la primera instancia del predicado `accion/3`, pues siempre se va a dar que el robot, la basura y la papelera estén en una posición.

También encontramos algún error menor de escritura, como por ejemplo en la línea 8, donde la variable `Pos3` hace una aparición "singleton". En su lugar debería de aparecer la variable `Pos1`, pues si el robot va a recoger la basura debería darse que ambos estuvieran en la misma posición.

Tras aplicar las mencionadas correcciones el código (que funciona) queda del siguiente modo:

```
1  % El estado se compone de tres posiciones, la posicion del robot, de
2  % la papelera y la basura.
3  estado(cuadricula(1,1),cuadricula(1,2), cuadricula(1,3)).
4
5  % El robot puede soltar la basura si esta cargada y estamos al lado de la papelera.
6  accion(estado(Pos1,Pos1, cargada), soltar, estado(Pos1, Pos1, en_papelera)).
7  % El robot puede recoger la basura si esta encima de ella.
8  accion(estado(Pos1,Pos2, Pos1), recoger, estado(Pos1, Pos2, cargada)).
```

```
% El robot puede moverse entre las distintas posiciones.
accion(estado(Pos1,Pos2, Pos3), ir(Pos1,Pos4), estado(Pos4, Pos2, Pos3)).

%% Los planes empiezan en un estado y acaban en otro.
plan(Estado, Estado, []). plan(Inicio, Fin, [Accion1|Resto]) :-
accion(Inicio, Accion1, Estado), plan(Estado,Fin,Resto).

%%:-plan(estado(cuadricula(1,1), cuadricula(1,2),cuadricula(1,3)),
↪  estado(_,_,en_papelera), Plan)
%%,write(Plan).
```

6) ***Suponed que la cuadricula es de cinco por cinco, donde las coordenadas son números enteros entre 1 y 5. Devolved una lista con los pasos que tiene que ir dando el robot para llegar a cada posición de la cuadrícula. Suponer que el robot solamente avanza de casilla en casilla, y se puede mover de la casilla cuadricula(x,y) a cuadricula(x+1,y), cuadricula(x-1,y), cuadricula(x,y+1), cuadricula(x,y-1).***

Para modelar el nuevo movimiento del robot (entre casillas adyacentes solo), hemos añadido el predicado `ir/3`, con el que podemos obtener las coordenadas de aquella casilla adyacente, que minimiza la función distancia con respecto a aquella casilla que se quiere alcanzar (si la basura está cargada esta será la de la papelera y en caso contrario será la de la basura).

Para la obtención de la lista de casillas adyacentes por las que hay que pasar para ir entre dos puntos de la cuadrícula hemos creado el predicado `camino/3` que nos devuelve mediante una consulta estos puntos en forma de lista. Dicha consulta la hemos incluido en los comentarios de final del código.

```
% El estado se compone de tres posiciones, la posicion del robot, de
% la papelera y la basura.
estado(cuadricula(1,1),cuadricula(1,2), cuadricula(1,3)).

ir(cuadricula(X1, Y1), cuadricula(X2, Y2), cuadricula(X3, Y3)):-
    abs(X3 - X1) > abs(Y3 - Y1),
    X2 is X1 + sign(X3 - X1),
    Y2 is Y1.

ir(cuadricula(X1, Y1), cuadricula(X2, Y2), cuadricula(X3, Y3)):-
    abs(Y3 - Y1) >= abs(X3 - X1),
    Y2 is Y1 + sign(Y3 - Y1),
    X2 is X1.

camino(cuadricula(X, Y), cuadricula(X, Y),[cuadricula(X , Y)]).

camino(cuadricula(X1, Y1), cuadricula(X2, Y2), [cuadricula(X1, Y1)|Tail]):-
    ir(cuadricula(X1, Y1), Pos, cuadricula(X2, Y2)),
    camino(Pos, cuadricula(X2, Y2), Tail),!.

% El robot puede soltar la basura si esta cargada y estamos al lado de la papelera.
accion(estado(Pos1, Pos1, cargada), soltar, estado(Pos1, Pos1, en_papelera)).
% El robot puede recoger la basura si esta encima de ella.
accion(estado(Pos1, Pos2, Pos1), recoger, estado(Pos1, Pos2, cargada)).
% El robot puede moverse entre las distintas posiciones.
accion(estado(Pos1, Pos2, cargada), ir(Pos1, Pos4), estado(Pos4, Pos2, cargada)):-
    ir(Pos1, Pos4, Pos2).
accion(estado(Pos1, Pos2, Pos3), ir(Pos1, Pos4), estado(Pos4, Pos2, Pos3)):-
    ir(Pos1, Pos4, Pos3).

%% Los planes empiezan en un y acaban en otro.
plan(Estado, Estado, []).

plan(Inicio, Fin, [Accion1|Resto]) :-
        accion(Inicio, Accion1, Estado),
    plan(Estado,Fin,Resto), !.
```

```
37 %%Consulta Plan:-
   ↪ plan(estado(cuadricula(1,1),cuadricula(1,2),cuadricula(1,3)),estado(_,_,en_papelera),Plan),write(Pl
38 %%Consulta Camino:- camino(cuadricula(3, 5), cuadricula(5, 3), Camino).
```

La gestión de errores de que las coordenadas estén entre $[1,5] \times [1,5]$ no la hemos realizado puesto que tampoco se realizaba en el código proporcionada aunque la habitación fuera de 3×3. Además dado que nos hemos cerciorado de que la casilla a la que se mueve el robot es la que minimiza la función distancia, podemos concluir que si las coordenadas iniciales del robot, la basura y la papelera están dentro de la habitación (5×5), el robot nunca saldrá de la habitación, pues en ningún caso las casillas de fuera van a estar en la ruta óptima entre dos puntos cualesquiera de la cuadrícula.

El problema con los horarios de autobuses

Se quiere realizar un viaje en autobús en Cantabria. Se dispone del siguiente horario:

```
1  :- op( 100, xfx, salida).
2  :- op( 50, xfx,:).
3  % Horario entre las siguientes localidades cantabras.
4  horario( [ santander salida 8:00, guarnizo salida 8:35, solares salida 8:55 ]).
5  % Empezamos en Santander salida 8:00, llegamos a guarnizo salida 8:35, seguimos a
   ↪ solares, salida 8:55 horario( [ santander salida 9:10, lierganes salida 9:25,
   ↪ guarnizo salida 9:55, solares salida 10:15 ]). horario( [ santander salida 9:45,
   ↪ lierganes salida 10:00, guarnizo salida 10:30, solares salida 10:50 ]). horario( [
   ↪ santander salida 11:45, lierganes salida 12:00, guarnizo salida 12:30, solares
   ↪ salida 12:50 ]). horario( [ santander salida 13:10, guarnizo salida 13:32, solares
   ↪ salida 13:45 ]).
6  horario( [ santander salida 14:05, guarnizo salida 14:40, solares salida 15:00 ]).
7  horario( [ santander salida 15:00, guarnizo salida 15:36, solares salida 15:57, beranga
   ↪ salida 16:13 ]). horario( [ santander salida 16:20, lierganes salida 16:35,
   ↪ guarnizo salida 17:05, solares salida 17:25 ]). horario( [ santander salida 18:05,
   ↪ lierganes salida 18:20, guarnizo salida 18:50, solares salida 19:10 ]). horario( [
   ↪ solares salida 9:00, guarnizo salida 9:20, lierganes salida 9:50, santander salida
   ↪ 10:05 ]). horario( [ solares salida 10:25, guarnizo salida 10:50, lierganes salida
   ↪ 11:20, santander salida 11:35 ]). horario( [ solares salida 11:25, guarnizo salida
   ↪ 11:45, santander salida 12:20 ]).
8  horario( [ beranga salida 12:55, solares salida 13:12, guarnizo salida 13:34, santander
   ↪ salida 14:10 ]). horario( [ solares salida 13:45, guarnizo salida 13:59, santander
   ↪ salida 14:20 ]).
9  horario( [ solares salida 15:05, guarnizo salida 15:25, santander salida 16:00 ]).
10 horario( [ solares salida 16:30, guarnizo salida 16:50, lierganes salida 17:20,
   ↪ santander salida 17:35 ]). horario( [ solares salida 18:15, guarnizo salida 18:35,
   ↪ lierganes salida 19:05, santander salida 19:20 ]). horario( [ solares salida 19:15,
   ↪ guarnizo salida 19:35, lierganes salida 20:05, santander salida 20:20 ]). %
   ↪ horario( StartPLace salida StartTime, EndPlace salida EndTime, Horario)
11 plan( Inicio, Destino, [ salida(Inicio), llegada( Siguiente) | Resto]) :-
12 list( Resto),
13 transbordo( Inicio, Siguiente), resto_horario(Siguiente ,Destino, Resto).
14 resto_horario( Sitio, Sitio, []).
15 resto_horario( Ahora, Destino, [ llegada( Siguiente) | Resto]) :- transbordo( Ahora,
   ↪ Siguiente),
16 resto_horario( Siguiente, Destino, Resto).
17 resto_horario( Sitio salida Hora1, Destino, [ esperar( Sitio, Hora1, Hora2) | Resto])
   ↪ :- transbordo( Sitio salida Hora2, _),
18 es_antes(Hora1, Hora2),
19 horario( Sitio salida Hora2, Destino, Resto).
20 transbordo( Sitio1, Sitio2) :- horario( List),
21 concatenar( _, [Sitio1,Sitio2|_ ], List). diferencia( H1:Min1, H2:Min2, Diff) :-
22 Diffis60*(H2-H1)+Min2-Min1.
```

```
es_antes(Hora1,Hora2):- diferencia(Hora1,Hora2,Diff), Diff>0.
list([]). dist([_|L]):-list(L).
concatenar([], L,L).
concatenar([X|L1], L2, [X|L3]):- concatenar(L1,L2,L3).
```

7) ***Corregir el programa para que funcione.***

Para hacer que el programa funcione se ha cambiado el orden de las distintas llamadas recursivas al predicado resto-horario() y se ha definido correctamente el predicado list(), obteniendose:

```
plan1( Inicio, Destino, [ salida(Inicio), llegada( Siguiente) | Resto]) :-
 list( Resto),
 transbordo( Inicio, Siguiente),
 resto_horario(Siguiente ,Destino, Resto).

resto_horario( Sitio, Sitio, []).
resto_horario( Sitio salida Hora1, Destino, [ esperar( Sitio, Hora1, Hora2) | Resto])
↪  :-
 transbordo( Sitio salida Hora2, _),
 es_antes(Hora1, Hora2),
 resto_horario( Sitio salida Hora2, Destino, Resto).
resto_horario( Ahora, Destino, [ llegada( Siguiente) | Resto]) :-
 transbordo( Ahora, Siguiente),
 resto_horario( Siguiente, Destino, Resto).

transbordo( Sitio1, Sitio2) :-
 horario( List),
 concatenar( _, [Sitio1,Sitio2|_ ], List).

diferencia( H1:Min1, H2:Min2, Diff) :-
 Diff is 60 * (H2 - H1) + Min2 - Min1.

es_antes(Hora1,Hora2):-
 diferencia(Hora1,Hora2,Diff), Diff>0.

list([]).
list([_|L]):-list(L).

concatenar([], L,L).
concatenar([X|L1], L2, [X|L3]):- concatenar(L1,L2,L3).
```

8) ***Desde Santander a las 9:10, ¿a que hora llegaremos a Solares? Poner la consulta que hay que hacer para responder a esta pregunta.***

```
?- plan1(santander salida 9:10, solares salida X, Plan).
X = 10:15,
Plan = [salida(santander salida 9:10), llegada(lierganes salida 9:25), llegada(guarnizo
↪  salida 9:55), llegada(solares salida 10:15)] .
```

9) ***Escribir una consulta para que nos de una ruta desde Beranga hasta Santander, estar en Santander 45 minutos y volver a Beranga en el mismo día.***

No es posible esperar en santander 45 minutos exactos debido a los horarios, pero si 50 minutos como en la consulta que se muestra a continuacion, una de las muchas combinaciones de irse de beranga y volver, pasando y esperando en santander:

```
?- plan1(beranga salida _, beranga salida _, Plan).
;
;
```

```
;
Plan = [salida(beranga salida 12:55), llegada(solares salida 13:12), llegada(guarnizo
↪  salida 13:34), llegada(santander salida 14:10), esperar(santander, 14:10, 15:0),
↪  llegada(guarnizo salida 15:36), llegada(solares salida ... : ...),
↪  llegada(...salida...)] .
```

10) ***¿Existe alguna forma salir de Santander, visitar Beranga y volver a Santander en el mismo día? ¿qué consulta hay que poner? ¿funciona esa consulta? Si no funciona, explicar porque y dad una forma de solucionarlo.***

No es posible ya que de beranga sale un unico tren por la mañana, y llega otro tren por la tarde, esto en todo el dia, por lo que en el mismo dia no es posible realizar el viaje deseado. Aun asi a continuacion se muestra la consulta que habria que realizar:

```
?- plan1(santander salida _, santander salida _, [_, beranga salida _|Plan]).
false.
```

la solucion seria añadir a la tabla de horarios un nuevo tren que saliera de beranga por las tardes, con lo que podriamos ir y volver en el mismo dia.