Taller de programación lógica

Verano 2019

Fecha límite de entrega: 11 de marzo de 2019 a las 16:00

1. Introducción

El *bondi* o colectivo es, con diferencia, el medio de transporte público más utilizado en la ciudad de Buenos Aires. Con 137 líneas distintas recorriendo la capital, a veces es difícil recordar cuáles de ellas nos llevan a nuestro destino, o dónde está la parada más cercana del bondi que nos queremos tomar. En este taller, usaremos Prolog para tratar de encontrar una solución a este problema.

Para esto contamos con una base de conocimiento (`paradas.pl`) que contiene la ubicación de una gran cantidad de paradas de colectivo a lo largo y a lo ancho de la ciudad.[1] La información de cada parada (su dirección y las líneas de bondi que paran allí) se representa a través de un *hecho*, siguiendo alguno de estos formatos:

- `parada(<calle>, <número>, <líneas>).`
- `paradaEsquina(<calle1>, <calle2>, <líneas>).`

donde los términos `<calle>`, `<calle1>` y `<calle2>` son *strings*, el término `<número>` es un número entero y el término `<líneas>` es una lista de números enteros.

Por ejemplo, el hecho

```
parada("GAONA AV.", 2659, [124, 84]).
```

indica que las líneas 124 (*Facultad de Derecho - Devoto*) y 84 (*Constitución - Villa del Parque*) paran en Av. Gaona 2659, mientras que

```
paradaEsquina("LAVALLE", "RODRIGUEZ PEÑA", [6]).
```

representa una parada de la línea 6 (*Villa Soldati - Retiro*) en la esquina de Lavalle y Rodríguez Peña.

Por otra parte, para hacer comparable la información entre estos dos formatos distintos, tenemos disponible una segunda base de conocimiento (`esquinas.pl`) que nos permite conocer a qué alturas corresponde la intersección entre dos calles. Por ejemplo, el hecho

```
esquina("JUSTO, JUAN B. AV.", 8000, "LOPE DE VEGA AV.", 600).
```

indica que en la esquina de las avenidas Juan B. Justo y Lope de Vega, la altura de la primera es 8000 y de la segunda, 600.

[1] Los datos proporcionados para este taller fueron obtenidos a partir de datasets públicos disponibles en https://data.buenosaires.gob.ar.

2. Ejercicios

Los ejemplos que acompañan cada ejercicio muestran los resultados esperados sobre el siguiente subconjunto de la base de conocimiento:

```
parada("ALEM, LEANDRO N. AV.", 638, [130, 93]).
parada("ALEM, LEANDRO N. AV.", 832, [132]).
parada("SANTA FE AV.", 4208, [15]).
parada("SANTA FE AV.", 4208, [55]).
parada("SANTA FE AV.", 4844, [15, 55]).
parada("FRAY JUSTO SANTAMARIA DE ORO", 2422, [111]).
parada("RIVADAVIA AV.", 5671, [132]).
parada("RIVADAVIA AV.", 5712, [104, 141, 36, 55, 88]).
paradaEsquina("ALEM, LEANDRO N. AV.", "RECONQUISTA", [132, 6, 7, 22]).
paradaEsquina("ALEM, LEANDRO N. AV.", "RECONQUISTA", [23, 26, 91, 28, 93]).
paradaEsquina("FRAY JUSTO SANTAMARIA DE ORO", "SANTA FE AV.", [64, 67, 57, 39, 152,
    41, 15, 93, 29]).
esquina("ALEM, LEANDRO N. AV.", 1150, "RECONQUISTA", 1200).
esquina("FRAY JUSTO SANTAMARIA DE ORO", 2500, "SANTA FE AV.", 4500).
```

> En los ejercicios del 1 al 4, considerar únicamente las paradas representadas según el predicado `parada` (es decir, ignorar las representadas mediante `paradaEsquina`).

Ejercicio 1

Dar consultas que permitan obtener la siguiente información a partir de la base de conocimiento:

(a) Dada una calle, todas los números donde hay una parada de colectivos y la lista de las líneas que paran allí.

(b) Dada una lista de líneas, todos los pares (`calle`, `número`) donde paran exactamente esas líneas.

Ejercicio 2

Definir un predicado `compartenParada(+Linea1, +Linea2, ?Calle, ?Numero)`, que sea verdadero si ambas líneas de colectivo (`Linea1` y `Linea2`) tienen una parada en `Calle` al `Numero`. Tener en cuenta que la base de conocimiento puede contener varios hechos para la misma calle y número.

Por ejemplo:

```
?- compartenParada(55, 15, C1, C2).
C1 = "SANTA FE AV.",
C2 = 4208 ;
C1 = "SANTA FE AV.",
C2 = 4844.
```

Ejercicio 3

Definir un predicado `viaje(?Linea, ?CalleO, ?NumO, ?CalleD, ?NumD)` que sea verdadero cuando la línea de colectivos `Linea` para tanto en la calle `CalleO` al `NumO` como en la calle `CalleD` al `NumD`. No debe ser verdadero si las direcciones de origen y de destino son idénticas

Útil: Predicado `diferentes/4`, que viene definido en el archivo `taller.pl`.

Por ejemplo:

```
?- viaje(55, "SANTA FE AV.", NO, CD, ND).
NO = 4208,
CD = "SANTA FE AV.",
ND = 4844 ;
NO = 4208,
CD = "RIVADAVIA AV.",
ND = 5712 ;
NO = 4844,
CD = "SANTA FE AV.",
ND = 4208 ;
NO = 4844,
CD = "RIVADAVIA AV.",
ND = 5712.

?- viaje(L, "ALEM, LEANDRO N. AV.", 832, C, N).
L = 132,
C = "RIVADAVIA AV.",
N = 5671.
```

Ejercicio 4

Definir los predicados:

(a) `lineasQueParan(+Calle, -Lineas)`, que, dada una `Calle`, sea verdadero para una lista `Lineas` que contenga exactamente todas las líneas de bondi con parada en esa calle, en cualquier orden y sin repetidos.

(b) `lineasQueParan(+Calle, +Numero, -Lineas)`, igual al inciso anterior pero en este caso debe especificar el número de calle.

Por ejemplo:

```
?- lineasQueParan("RIVADAVIA AV.", Ls).
Ls = [36, 55, 88, 104, 132, 141].
?- lineasQueParan("SANTA FE AV.", 4208, L)
Ls = [15, 55].
```

Útil: Metapredicado `setof(+Template, +Goal, -Set)`.

Ejercicio 5

Definir las reglas de inferencia necesarias para que el predicado `parada` también sea verdadero para las paradas que se encuentran en una esquina. El predicado debe ser completamente reversible también en estos casos.

Algunos ejemplos:

```
?- parada("SANTA FE AV.", 4500, Ls).
Ls = [64, 67, 57, 39, 152, 41, 15, 93, 29].

?- parada("FRAY JUSTO SANTAMARIA DE ORO", N, _).
N = 2422 ;
N = 2500.

?- parada(C, N, [132, 6, 7, 22]).
C = "ALEM, LEANDRO N. AV.",
```

```
N = 1150 ;
C = "RECONQUISTA",
N = 1200.
```

Ejercicio 6

Definir un predicado `paradaCercana(+Calle, +Numero, +Distancia, ?Parada)` que, dada una dirección, sea verdadero si `Parada` es un término que representa una parada sobre la misma calle, o se encuentra a la vuelta de la esquina. Estas se deben encontrar a una distancia (medida según la numeración de la calle) no mayor que la indicada.

Por ejemplo:

```
?- paradaCercana("ALEM, LEANDRO N. AV.", 700, 150, P).
P = parada("ALEM, LEANDRO N. AV.", 638, [130, 93]) ;
P = parada("ALEM, LEANDRO N. AV.", 832, [132]).

?- paradaCercana("SANTA FE AV.", 4400, 200, P).
P = parada("SANTA FE AV.", 4208, [15]) ;
P = parada("SANTA FE AV.", 4208, [55]) ;
P = parada("SANTA FE AV.", 4500, [64, 67, 57, 39, 152, 41, 15, 93|...]) ;
P = parada("FRAY JUSTO SANTAMARIA DE ORO", 2422, [111]) ;
P = parada("FRAY JUSTO SANTAMARIA DE ORO", 2500, [64, 67, 57, 39, 152, 41,
   15, 93|...]).
```

(aquí se considera que la distancia entre Av. Santa Fe 4400 y la parada del 111 que está sobre la calle Oro es 178, ya que para ir de un punto al otro, hay que recorrer 100 números sobre Av. Santa Fe y 78 sobre Oro).

Ejercicio 7

Un recorrido será para nosotros una lista de viajes, cada uno de ellos realizado en una línea de colectivos y representado por un término de la forma `viaje(Linea, CalleO, NumO, CalleD , NumD)`.

Definir un predicado `pasaPor(+Recorrido, ?Calle, ?Numero)` que sea verdadero si:

- alguna de las paradas visitadas en el `Recorrido` está en la dirección `Calle` al `Numero`, o bien
- alguno de los viajes que conforman el recorrido es un tramo recto que comprende dicha dirección; es decir, las paradas de los dos extremos del viaje están sobre `Calle`, mientras que `Numero` está comprendido entre los números de las dos paradas.

Por ejemplo:

```
?- pasaPor([viaje(132, "ALEM, LEANDRO N. AV", 832, "RIVADAVIA AV.", 5671),
    viaje(55, "RIVADAVIA AV.", 5712, "SANTA FE AV.", 4208)], "RIVADAVIA AV.",
     N).
N = 5712.

?- pasaPor([viaje(93, "ALEM, LEANDRO N. AV.", 638, "ALEM, LEANDRO N. AV.",
    1150)], C, N).
C = "ALEM, LEANDRO N. AV.",
N = 638 ;
C = "ALEM, LEANDRO N. AV.",
N = 1150 ;
```

```
C = "ALEM, LEANDRO N. AV.",
N = 639 ;
C = "ALEM, LEANDRO N. AV.",
N = 640 ;
C = "ALEM, LEANDRO N. AV.",
N = 641 ;
...
```

Útil: Predicado `between(+Inf, +Sup, ?Valor)`.

Ejercicio 8

Definir un predicado `recorrido(+CalleOrig, +NumOrig, +CalleDest, +NumDest, +Dist, +CantTrasbordos, -Recorrido)` que instancie `Recorrido` en una lista de viajes consecutivos que pueden realizarse para llegar desde la dirección de origen hasta la dirección de destino.

Los recorridos generados no deben implicar cambiar de colectivo más de `CantTrasbordos` veces, y la distancia entre la parada de destino de un viaje y la parada de origen del siguiente no debe ser mayor que `Dist`.

No se deberán generar recorridos en los que:

- se realice más de un viaje en la misma línea de colectivos,
- se pase más de una vez por una parada (tener en cuenta el predicado `pasaPor/3` definido en el ejercicio anterior).

Por ejemplo:

```
?- recorrido("SANTA FE AV.", 4100, "ALEM, LEANDRO N. AV.", 800, 200, 1, R).
R = [
    viaje(15, "SANTA FE AV.", 4208, "FRAY JUSTO SANTAMARIA DE ORO", 2500),
    viaje(93, "FRAY JUSTO SANTAMARIA DE ORO", 2500, "ALEM, LEANDRO N. AV.",
   638)
] ;
R = [
    viaje(15, "SANTA FE AV.", 4208, "FRAY JUSTO SANTAMARIA DE ORO", 2500),
    viaje(93, "SANTA FE AV.", 4500, "ALEM, LEANDRO N. AV.", 638)
] ;
R = [
    viaje(15, "SANTA FE AV.", 4208, "SANTA FE AV.", 4500),
    viaje(93, "SANTA FE AV.", 4500, "ALEM, LEANDRO N. AV.", 638)
] ;
R = [
    viaje(15, "SANTA FE AV.", 4208, "SANTA FE AV.", 4500),
    viaje(93, "FRAY JUSTO SANTAMARIA DE ORO", 2500, "ALEM, LEANDRO N. AV.",
   638)
] ;
R = [
    viaje(55, "SANTA FE AV.", 4208, "RIVADAVIA AV.", 5712),
    viaje(132, "RIVADAVIA AV.", 5671, "ALEM, LEANDRO N. AV.", 832)
].
```

3. Condiciones de evaluación

Los objetivos a evaluar con este taller son:

- Corrección.
- Declaratividad.
- Prolijidad. Se deberá evitar repetir código innecesariamente y usar adecuadamente los predicados previamente definidos.
- Uso adecuado de unificación, backtracking, generate and test y reversibilidad de los predicados que correspondan.

A menos que se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución (no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada).

Las sugerencias de los ejercicios pueden ayudar, pero no es obligatorio seguirlas. Pueden escribirse todos los predicados auxiliares que se requieran, aclarando siempre de manera explícita cuáles de los argumentos deben estar instanciados (usando `+`, `-` y `?`).

4. Pautas de entrega

Recomendamos encarecidamente aprovechar las clases destinadas al taller para intentar terminar los ejercicios.

Aquellos grupos que no cuenten con las correcciones al finalizar la clase del 7 de marzo, deberán:

- entregar el código antes de la fecha límite de entrega utilizando el Campus Virtual. Para hacerlo, uno de los dos integrantes del grupo deberá ingresar a la tarea "Taller de programación lógica" en la sección "Talleres" de la página del Campus y subir allí el código.
- entregar una versión impresa del código.

Importante:

- El código entregado debe poder ser ejecutado en SWI-Prolog.
- El código debe incluir tests que permitan probar los predicados pedidos.
- No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código).
- Para poder hacer el envío, el grupo debe estar cargado en el Campus. En caso de que la composición del grupo sea diferente a la del taller anterior, les solicitamos que informen a los docentes antes de la fecha límite de entrega, para poder hacer la actualización en el Campus.
- El sistema dejará de aceptar envíos pasada la fecha límite de entrega y no se harán excepciones. Por favor, planifiquen el trabajo para llegar a tiempo con la entrega.

Algunos predicados y metapredicados definidos en SWI-Prolog

Esta sección contiene algunos predicados y metapredicados ya definidos en la actual implementación de SWI-Prolog que pueden ser de utilidad para el desarrollo del taller. La descripción de cada uno se puede hallar en la ayuda de SWI-Prolog (invocada con el predicado `help`).

Recordar que en algunos ejercicios puede ser conveniente definir el predicado opuesto al que se pide en el enunciado, y luego usar **`not`**. En este caso, tener especial cuidado con la instanciación de las variables.

- Predicados sobre listas:
  - `append(-List1, -List2, -List3).`
  - `maplist(+Pred, -List)`
  - `maplist(+Pred, -List1, -List2)`
  - `maplist(+Pred, -List1, -List2, -List3)`
  - `member(-Elem, -List)`
  - `nth0(-N, -List, -Elem)`
  - `select(-Elem, -List, -Rest)`
  - `sublist(+Pred, +List1, -List2)`
  - `subset(+Subset, -Set)`
- Metapredicados:
  - `forall(+Cond, +Action)`
  - **`not`**`(+Goal)`
  - `setof(+Template, +Goal, -Set)`