Lógica Computacional
Práctica 05: Programación Lógica

Dicter Tadeo García Rosas Erik Rangel Limón Manuel Soto Romero

Semestre 2024-2
Facultad de Ciencias UNAM

Objetivos

1. Entender los conceptos de la programación lógica con el lenguaje de programación *Prolog*.

Instrucciones

1. Para esta práctica necesitan tener instalado *Prolog*.

   - *Linux:*
     Por lo general el paquete que necesitan tiene el nombre `swi-prolog`, investiguen si esto difiere según su distribución.
     Debian:

     ```
     sudo apt install swi-prolog-full
     ```

     Fedora:

     ```
     sudo dnf install pl
     ```

   - *Mac:*

     ```
     brew install swi-prolog
     ```

   - *Windows:*
     En la página `https://www.swi-prolog.org/download/stable` descargar la versión correspondiente con su sistema.

2. Resolver todas las funciones que se encuentran en el archivo `src/plogica.pl`.

   Su calificación será decidida manualmente, se verificará que los ejercicios cumplan con la especificación y se verificará la calidad del código.

Ejercicios

1. *Buen fin*:

   Dada la siguiente información, define los hechos, reglas y consultas necesarias para encontrar las 3 tuplas (Nombre, Prenda, Color, Descuento) que resuelven el acertijo:

   Curry, Kleene y Russell fieron de compras durante el Buen Fin para aprovechar las rebajas. Cada uno compró un artículo y tuvo su debido descuento. Los artículos que compraron son una chamarra, un pantalón y una sombrilla; y encontraron descuentos del 10 %, 25 % y hasta 50 % con meses sin intereses. Tenemos las siguientes pistas:

   - *La chamarra azul no la compró Curry.*
   - *Russell estuvo alegre de encontrar un 25 % de descuento en el producto que compró.*
   - *El producto rojo tuvo un mayor descuento que la sombrilla.*
   - *Los pantalones no tuvieron el descuento del 10 % y no los compraron ni Curry ni Kleene.*

   Nota importante: En el README deberán especificar qué consulta nos da las 3 tuplas buscadas.

2. *Partición:*

   Define el predicado `particion(L1, L2, L3)` que se satisface cuando L2 es la primera mitad de la lista L1, y L3 es la segunda mitad de L1. Si el número de elementos es impar, una lista puede quedar más grande que la otra por un elemento.

   Ejemplos:

```
?- particion([1,2,3,4,5,6,7], L, R).
L = [1, 2, 3],
R = [4, 5, 6, 7].
```

3. *Combina:*

   Define el predicado `combina(L1, L2, L3)` que se satisface cuando dadas dos listas L1 y L2 ordenadas de forma ascendente, L3 es la lista ordenada con los elementos de las primeras dos

   Ejemplo:

```
?- combina([2, 4, 6], [1, 3, 5, 7], L).
L = [1, 2, 3, 4, 5, 6, 7].
```

4. *Máximo común divisor*:

   Define el predicado `gcd(X, Y, Z)` que se satisface cuando Z es el maximo común divisor de X y Y. Ademas, define `coprimos(X, Y)` que decide si X y Y son primos relativos.

   Ejemplos:

```
?- gcd(13, 19, X).
X = 1.
?- coprimos(13,19).
true.
?- gcd(24, 18, X).
X = 6.
?- coprimos(24,18).
false.
```

5. *Agrupación:*

   Define el predicado `agrupa(XS,XXS)` donde XXS una lista de listas que contenga los elementos de XS agrupados de tal forma que los elementos iguales consecutivos se encuentran en la misma sublista en el orden en el que aparecieron.

   Ejemplos:

```
?- agrupa([a,a,b,b,c,c,c,d,1,1],X).
X = [[a, a], [b, b], [c, c, c], [d], [1, 1]].
?- agrupa([1,2,3,4,5],X).
X = [[1], [2], [3], [4], [5]].
?- agrupa([1,1,7,2,2,2,2,7,1],X).
X = [[1, 1], [7], [2, 2, 2, 2], [7], [1]].
```

6. *Pertenencia:*

   Define el predicado `pertenece(X,XS)` que nos dice si X es un elemento que aparece en la lista XS.

   Ejemplos:

```
?- pertenece(1,[5,1,a,c,x]).
true.
?- pertenece(a,[a,a,a,a,a,a]).
true.
?- pertenece(e,[a,b,c,d]).
false.
```

7. *Elimina:*

   Define el predicado `elimina(E,XS,YS)` donde YS sea la lista resultado de eliminar la primera aparición de E en XS.

   Ejemplos:

```
?- elimina(1,[5,1,4,3,2,1],Y).
Y = [5,4,3,2,1].
?- elimina(a,[c,f,e,t,g],Y).
Y = [c,f,e,t,g].
```

8. *Caminos:*

   Toma en cuenta el siguiente mapa:

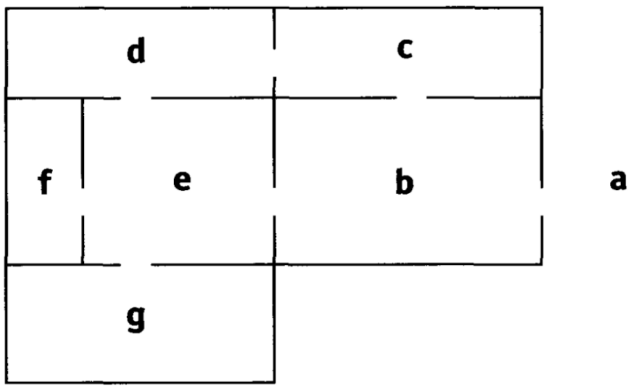

Figura 1: Mapa de una casa

.

Da un predicado `camino(X,Y,XS)` donde XS es un camino que va de X a Y sin repetir cuartos.

Define la base de conocimientos necesaria para resolver el problema.

```
? - camino (a, d, P).
P = [a, b, c, e] ;
P = [a, b, d, e].
```

Entrega

1. La tarea se entrega en *Github Classroom*.
2. La tarea se entrega en equipos de hasta 4 personas.
3. Además de resolver los ejercicios, deben entregar un `README` (`.md` o `.org`) en la raiz de la carpeta de la práctica que contenga los datos de todos los integrantes, y por cada una de sus funciones dejar una breve explicación de su solución.

Consideraciones

1. Todas las prácticas con copias totales o parciales tanto en el código como en el `README` serán evaluadas con cero.
2. Las únicas funciones con soluciones iguales admisibles son todas aquellas que sean iguales a las resueltas por el grupo en el laboratorio, sin embargo la explicación de su solución en el `README` debe ser única para cada equipo.
3. No entregar el `README` o tenerlo incompleto se penalizará con hasta dos puntos menos sobre su calificación en la práctica.
4. Cada día de retraso se penalizará con un punto sobre la calificación de la práctica.
5. No se recibirán prácticas que no compilen (no debe arrojar errores la orden `swipl plogica.pl`).
6. Las participaciones en el laboratorio se aplicarán de manera individual sobre la calificación que tuvieron en la práctica.