Facultad de Ciencias - UNAM
Lógica Computacional 2023-1
Práctica 4 - Introducción a Prolog

Favio E Miranda Perea
Javier Enríquez Mendoza
José Manuel Madrigal Ramírez

8/noviembre/2022
Fecha de entrega: 15/noviembre/2022 a las 23:59 hrs

1. Introducción

Prolog es un lenguaje de programación declarativo que fue desarrollado en la universidad de Marselle por Alain Colmenrauer y Philippe Roussel en 1972. Se ha utilizado en la demostración de teoremas, en la implementación de sistemas expertos, en el procesamiento del lenguaje natural, entre otras aplicaciones. El objetivo de esta práctica es abordar los conceptos básicos de este lenguaje:

- Sintaxis básica y bases de conocimiento
- Parentesco, árboles SLD y grafos
- Aplicación en números naturales
- Listas

Para comenzar, vamos a instalar SWI-Prolog, una implementación en código abierto del lenguaje:

Instrucciones de instalación para distintos sistemas operativos

Sintaxis básica y bases de conocimiento

Al ejecutar SWI-Prolog se desplegará una terminal desde la que podremos consultar nuestras bases de conocimiento. Una base de conocimiento es un archivo `.pl` como el que subí junto a este PDF. Basta escribir el comando `consult` en la terminal de SWI-Prolog y la dirección de nuestra base para poder interactuar con ella, veamos:

```
consult(c:/Users/jose/Documents/Prolog/base1.pl).
true.
```

También podemos consultar la base utilizando el menú en la interfaz de SWI-Prolog, pero cuando depuramos un programa es más eficiente utilizar las flechas del teclado para navegar hacia el último comando de consulta que hayamos ejecutado y volver a cargar nuestra base.

En una base de conocimientos encontraremos enunciados de dos tipos:

- Hechos
- Reglas

Dentro de estos enunciados figuran variables y objetos. Las variables siempre se escribirán comenzando con letra mayúscula o guión bajo:

`_var1` `Objeto2` `Resultado` `X` `_23`

y los objetos comenzando con letra minúscula.

`elefante` numeron1 `24` `blacky` `elemento1`

Los **hechos** se componen por el identificador de una relación (también llamado functor) y un conjunto de objetos entre paréntesis:

```
es_perro( blacky).
mas_grande(elefante, caballo).
```

Y al consultar estos hechos en la terminal de SWI-Prolog obtenemos:

```
?- es_perro(blacky).
true.
?- es_perro(rufus).
false.
?- mas_grande(elefante, caballo).
true.
?- mas_grande(caballo, caballo).
false.
```

Podemos utilizar variables en la terminal para preguntar cosas sobre nuestra base de conocimiento. Por ejemplo, podemos preguntar por los objetos que cumplen con el predicado `es_gato` de la siguiente manera:

```
?- es_gato(X).
X = masapan;
X = waffle;
X = curie;
X = crepita.

?-
```

Observa cómo utilizamos el punto y coma al final de cada respuesta para averiguar si hay otro objeto que cumpla con el predicado en cuestión. Cuando no existan más respuestas SWI-Prolog desplegará el prompt `?-` y añadirá punto final a nuestra instrucción. Prueba ejecutando algunos enunciados de este estilo en la terminal.

Por otro lado, las **reglas** se componen de una cabeza y un cuerpo separados por el símbolo **:-**, como sucede en el siguiente ejemplo:

```
mucho_mas_grande(X,Y):- mas_grande(X, Y).
mucho_mas_grande(X,Y) :- mas_grande(X, Z), mucho_mas_grande(Z, Y).
```

La cabeza tiene una sintaxis similar a los hechos mientras que el cuerpo se compone de uno o más enunciados separados por comas o puntos y coma.

Veamos algunos ejemplos de su ejecución en terminal (seguimos utilizando la base de conocimientos inicial que acompaña esta práctica y que contiene varios hechos con el predicado `mas_grande`):

```
?- mas_grande(caballo,X).
X = perro.
?- mucho_mas_grande(caballo,X)
X = perro;
X = raton;
X = hormiga;
false.

?-
```

Podemos pensar en las reglas de Prolog como enunciados de implicación en los que el cuerpo representa el antecedente y la cabeza el consecuente. Si el cuerpo es verdadero la cabeza es verdadera.

Cabe aclarar que las comas representan conjunciones y los puntos y coma representan disyunciones. En el ejemplo de la regla **mucho_mas_grande** cabe notar que de definirla al revés, poniendo la parte autoreferencial del caso recursivo, es decir, escribiendo:

```
mucho_mas_grande(X,Y) :- mucho_mas_grande(Z, Y),mas_grande(X, Z).
```

en vez de:

```
mucho_mas_grande(X,Y) :- mas_grande(X, Z), mucho_mas_grande(Z, Y).
```

tendríamos un error por desbordameinto de pila debido a la manera en que Prolog hace backtracking y selecciona las cláusulas a ejecutar de manera interna. De modo que siempre es mejor escribir la parte autoreferencial en los casos recursivos de nuestras reglas al final de las cláusulas.

Finalmente no debemos olvidar que toda línea de código llevará un punto final.

Puedes consultar el siguiente enlace en caso de querer profundizar más en la sintaxis básica del lenguaje:

Conceptos básicos de Prolog

Ejercicios

Parentesco, árboles SLD y grafos

1. En el archivo *basedc.pl* hay un conjunto de hechos sobre el parentesco de algunas personas. El predicado `progenitorde(X,Y)` se considera verdadero si X es progenitor de Y.

   - Escribe una regla `eshijode(X,Y)` que sea verdadera si X es, en efecto, hijo de Y.
   - Escribe otra regla que sea verdadera si X es abuelo de Y. Elige un identificador conveniente para la misma.
   - Escribe una regla `eshermanode(X,Y)` que sea verdadera si X es hermano o hermana de Y.
   - Escribe el cuerpo de una regla que indique si entre dos objetos dados hay parentesco. Considera las relaciones de hermanos, progenitores y ancestros. Elige un identificador conveniente para esta regla.

2. (**Extra**) Considera la siguiente definición para la suma de los primeros N naturales:

   ```
   suma(0, 0).
   suma(X,Res) :- Y is X - 1,suma(Y,S), Res is  X + S.
   ```

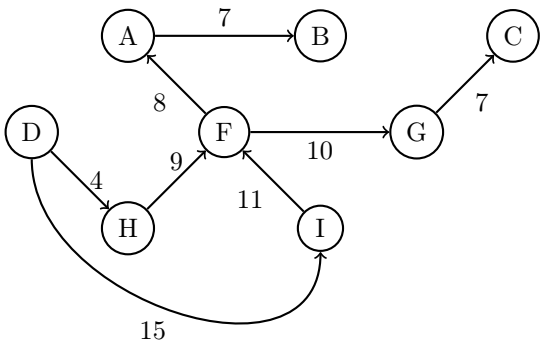
   Transcríbela en la base de datos y realiza algunas consultas. Después investiga el desarrollo de los *árboles de selección lineal de cláusulas definidas*, también llamados árboles SLD y desarrolla el árbol para la siguiente consulta:

   ```
   ?- suma(2,R).
   ```

   Observa la relevancia del operador **is** en la definición de `suma`. Este operador nos sirve en contextos similares donde queramos unificar el resultado de una operación aritmética con alguna variable dentro de la cabeza o el mismo cuerpo de la regla.

```
?-5 is 3 + 2.
true.
?-35 is 7 * 5.
true.
```

3. Considera la siguiente gráfica:



- Escribe un conjunto de hechos en la base de conocimiento que representen las relaciones entre los nodos y su costo correspondiente. Aquí algunos ejemplos:

```
?-arista(d,h,X).
X = 4.
?-arista(h,c,X).
false.
```

- Escribe el cuerpo de una regla cuya cabeza sea verdadera si existe un camino entre dos nodos dados:

```
?-camino(d,a).
true.
?-camino(f,i).
false.
```

- Escribe una regla que indique el costo de recorrer un camino entre dos nodos si es que existe:

```
?-costo(d,h,X).
X = 4.
?-costo(h,b,X).
X = 24.
?-costo(b,h,X).
false.
```

Números naturales

Veamos la siguiente definición para los números naturales:

```
esNatural(c).
esNatural(s(X)) :- esNatural(X).
```

Transcríbela en la base de conocimiento y después escribe el cuerpo de las siguientes reglas:

- `suma(X, Y, Z)`, es verdadera si Z es la suma de X y Y. Las tres variables deben ser naturales escritos como conjuntos que contienen al cero.

```
?-suma(s(s(c)), s(c), X).
X = s(s(s(c))).
?-suma(t, s(c)).
false.
```

- `multiplicación(X, Y, Z)`, es verdadera si Z es el producto de X y Y.

```
?-multiplicacion(s(s(s(c)))), s(s(c)), X).
X = s(s(s(s(s(c))))).
?-multiplicacion(s(c), estavariablenoesunnatural).
false.
```

- `aDecimal(N, Z)`, es verdadera si Z es la versión en decimal de N. Por ejemplo:

```
?-a_Decimal(s(s(s(s(s(c))))),X)
X = 6.
?-a_Decimal(s(t(s(c)))).
false.
```

Listas

Aprendamos la sintaxis de las listas en Prolog con el siguiente ejemplo:

```
listar(L) :- Xs = [2,3,4,5], H = 1, L = [H|Xs].
```

Al llamar la regla anterior desde la terminal de SWI-Prolog obtenemos:

```
?- listar(X).
?- X = [1,2,3,4,5].
```

Como podemos ver, las listas están compuestas por una cabeza separada y una lista (la cola o resto de la lista original), separadas por un pipe: |.

Escribamos una función que nos devuelva la cabeza de una lista:

```
cabeza([C|L],Cabeza) :- Cabeza is C .
```

Pero en aras de mantener las cosas simples, reescribámosla en forma de un hecho:

```
cabeza([C|L],C).
```

En ambos casos, al ejecutar la pregunta desde la terminal obtendremos:

```
?- cabeza([1,2,3,4], X).
X = 1.
?- caheza([1,2,3,4], 2.
false.
```

Ejercicios

1. Escribe un hecho que regrese una lista sin su cabeza:

   ```
   ?- sinCabeza([1,2,3,4], X).
   X = [2,3,4].
   ```

2. Copia el siguiente par de expresiones en la base de conocimientos:

   ```
   ancestro(pedro, [ana, ramon]).
   ancestro(ana, [ramon, pepe, juan]).
   ```

   Escribe el predicado `descendienteP(X,Y).` que es verdadero si `Y` es el primer miembro en la lista de descendientes de `X`. Por ejemplo:

   ```
   ?- descendienteP(ramon, X).
   X = ana.
   ?-descendienteP(ramon, juan.
   false.
   ```

3. (**Extra**) Considerando también las siguientes listas:

   ```
   ancestro(pedro, [ana, ramon, pedro, javier, vilma, nicolas]).
   ancestro(juan, [ben, pepe, josue, jesica, pavel, keith, kyle]).
   ```

   Escribe una regla `descendiente(X,Y)` que es verdadera si X es descendiente de Y.

   ```
   ?- descendiente(nicolas, X).
   X = pedro.
   ?-descendiente(ramon, dafne).
   false.
   ```

Pista: Escribe una regla auxiliar de búsqueda en listas. Cuando la lista es vacía, un caso base se ve de esta manera:

```
buscar(elem, []) ;- !, fail.
```

El signo de admiración sirve para podar ramas del árbol SLD innecesarias. Prueba escribiendo el caso base restante y el caso recursivo en clausulas aparte. El símbolo para comparar == puede ayudarte. Después utiliza esta regla como apoyo para escribir la regla `descendiente`.

Entrega

La entrega se realizará por **parejas** y consistirá en un archivo comprimido que debe contener:

- El archivo `basedc.pl` con el código necesario que da solución a los ejercicios.
- Un archivo readMe.txt que incluya sus nombres y una bitácora sobre el desarrollo de la práctica donde indiquen los problemas que se presentaron y cómo los resolvieron. Asimismo incluyan comentarios de cada uno al respecto.

Envíen su archivo comprimido a mi correo: *jose.manuel.madrigal.ramirez@gmail.com* con el siguiente formato:

Practica4-Apellido1Nombre1Apellido2Nombre2.zip

Por ejemplo, si su equipo esta conformado por Alan Turing y Ada Lovelace su archivo debería llamarse:

Practica4-TuringAlanLovelaceAda.zip

Cualquier duda que tengan no duden en enviarme un correo o comunicarse por Telegram. 😄