Universidad Veracruzana

Inteligencia Artificial

Programación para la Inteligencia Artificial

Actividad 2

Leonardo Flores Torres

26 de octubre de 2022

1. Programe en Prolog un algoritmo de unificación. A reportar **(50 puntos)**:
   - *a*) el algoritmo elegido comentado;
   - *b*) su código, también comentado;
   - *c*) Los siguientes ejemplos de la ejecución:
     1) $q(Y, g(a, b))$, $p(g(X, X), Y)$.
     2) $r(a, b, c)$, $r(X, Y, Z)$.
     3) $mayor(padre(Y), Y)$, $mayor(padre(Z), juan)$.
     4) $conoce(padre(X), X)$, $conoce(W, W)$.
2. Implemente las siguientes operaciones sobre conjuntos representados como listas **(20 puntos)**:
   - Subconjunto:

```
?- subset([1,3], [1,2,3,4]).
true.
?- subset([], [1,2]).
true.
```

   - Intersección:

```
?- inter([1,2,3], [2,3,4], L).
L = [2, 3].
```

   - Unión:

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

   - Diferencia:

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

3. Escriba un predicado que convierta números naturales de Peano a su equivalente decimal. Posteriormente implemente la suma y la resta entre dos números de Peano **(10 puntos)**. Por ejemplo:

```
?- peanoToNat(s(s(s(0))), N).
N = 3.
?- peanoToNat(0, N).
N = 0.
?- sumaPeano(s(s(0)), s(0), R).
R = s(s(s(0))).
?- restaPeano(s(s(0)), s(0), R).
R = s(0).
```

Solución:

Primero definí una manera de convertir números naturales en la representación de Peano:

```
%%% natToPeano/2 Computes the conversion of natural numbers to their Peano
↪  representation
%%% Ejemplo:
% ?- natToPeano(3, X).
% X = s(s(s(0))) .
natToPeano(0, 0).
natToPeano(N, s(X)) :-
    N > 0,
    N1 is N - 1,
    natToPeano(N1, X).
```

Con la regla `natToPeano/2` traté de hacer otra regla `peanoToNat/2` para encontrar el número natural correspondiente a un número de Peano pero no obtenia resultados satisfactorios:

```
peanoToNat(P, N) :-
    natToPeano(N, P).
```

Por lo que definí una regla distinta para este inciso:

```
%%% peanoToNat/2 computes the respective natural number of a Peano number
%%% Example:
% ?- peanoToNat(s(s(s(0))), X).
% X = 3.
peanoToNat(0, 0).
```

```
peanoToNat(s(X), N) :-
    peanoToNat(X, N1),
    N is N1 + 1.
```

Recordando un poco acerca de los comentarios hechos en clase sobre la librería de dominios finitos decidí utilizarla solo para hacer una definición alternativa de las dos reglas anteriormente definidas:

```
%%% With finite domains
%%% natToPeano_fd/2 computes the conversion of a natural number to its Peano
↪ representation
natToPeano_fd(0, 0).
natToPeano_fd(N, s(X)) :-
    N #> 0,
    N1 #= N - 1,
    natToPeano_fd(N1, X).
% peanoToNat_fd/2 computes the conversion of a Peano number to its natural number
↪ equivalent
peanoToNat_fd(S, N) :-
    natToPeano_fd(N, S).
```

De esta manera sí puedo utilizar `natToPeano_fd/2` para definir `peanoToNat_fd/2`. Cabe mencionar que el resto de las operaciones requeridad para este inciso se elaboraron tomando las reglas que no utilizan la librería de dominio finito.

La operación de suma se implementó con la siguiente regla:

```
%%% addPeano/3 adds S2 to S1, S3 = S1 + S2
addPeano(S1, S2, S3) :-
    peanoToNat(S1, N1),
    peanoToNat(S2, N2),
    N3 is N1 + N2,
    natToPeano(N3, S3).
```

---

4. Escriban un predicado `pino/1` cuyo argumento es un entero positivo y su salida es como sigue **(10 puntos)**:

```
?- pino(5).
    *
   * *
  * * *
 * * * *
* * * * *
true.
```

Solución:

El programa para solucionar este inciso se muestra a continuación:

```
% pine/1 creates a pine on a desired number of levels specified by Levels
pine(Levels) :- pine(0, Levels), !.

% pine/2 iterates through the lines appending spaces each line to center the
↪  stars
pine(C, X) :- C < X,
    C1 is C+1,
    Y1 is X - C1,
    spaces(Y1),
    stars(0, C),
    pine(C1, X).
pine(C, X) :- C >= X.

% spaces/1 writes N spaces on demand
spaces(0) :- write('').
spaces(N) :-
    N1 is N - 1,
    N1 >= 0,
    write(' '),
    spaces(N1).

% stars/2 writes stars on demand on one line only and writes a new line after it
↪  has finished
stars(X, Y) :- X =< Y,
    X1 is X + 1,
    write('* '),
    stars(X1, Y).
stars(X, Y) :- X > Y, nl.
```

Haciendo un query en Prolog para un pino de 7 pisos da el siguien resultado:

```
?- pine(7).
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *
true.
```

Jugando un poco con el programa se puede hacer un pino borracho:

```
?- pine(7).
      *
     *    *
    *    *    *
   *    *    *    *
  *    *    *    *    *
 *    *    *    *    *    *
*    *    *    *    *    *    *
true.
```

5. Escriba un programa que regrese en su segundo argumento la lista de todas las permutaciones de la lista que es su primer argumento **(10 puntos)**. Por ejemplo:

```
?- perms([1,2,3], L).
L = [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1],
     [3, 1, 2], [3, 2, 1]].
```

Solución:

Se hizo uso de `findall/3` y `permutation/2` definidas en la librería estándar de Prolog para realizar este ejercicio. Donde `findall/3` crea una lista de todas las instancias donde una meta definida tenga éxito, en este caso la meta es `permutation/2`.

```
%%% perms/2 computes the all possible permutations of List in Perms.
%%% Ejemplo:
% ?- perms([1,2,3], L).
% L = [[1, 2, 3], [1, 3, 2], [2, 1, 3],
%      [3, 1, 2], [2, 3, 1], [3, 2, 1]].
perms(List, Perms) :-
    findall(Perm, permutation(Perm, List), Perms).
```

Se mostraran un par de ejemplos de uso aparte del mostrado en el enunciado de la tarea el cual es incluido dentro de los comentarios de la solución:

```
?- perms([1,2], L); true.
L = [[1, 2], [2, 1]] .
?- perms([1,2,3], L).
L = [[1, 2, 3], [1, 3, 2], [2, 1, 3], [3, 1, 2], [2, 3, 1], [3, 2, 1]].
?- perms([1,2,3,4], L); true.
L = [[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 4, 2, 3], [1, 3, 4, 2], [1, 4,
↪  3|...], [2, 1|...], [2|...], [...|...]|...] [write]
L = [[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 4, 2, 3], [1, 3, 4, 2], [1, 4,
↪  3, 2], [2, 1, 3, 4], [2, 1, 4, 3], [3, 1, 2, 4], [4, 1, 2, 3], [3, 1, 4, 2],
↪  [4, 1, 3, 2], [2, 3, 1, 4], [2, 4, 1, 3], [3, 2, 1, 4], [4, 2, 1, 3], [3, 4,
↪  1, 2], [4, 3, 1, 2], [2, 3, 4, 1], [2, 4, 3, 1], [3, 2, 4, 1], [4, 2, 3, 1],
↪  [3, 4, 2, 1], [4, 3, 2, 1]] .
```

Para la última meta se especificó como `perms([1,2,3,4], L); true.` para poder tener la oportunidad de escribir `w` y mostrar la lista completa de permutaciones.