Universidad Veracruzana

Inteligencia Artificial

Programación para la Inteligencia Artificial

Actividad 2

Leonardo Flores Torres

27 de octubre de 2022

1. Programe en Prolog un algoritmo de unificación. A reportar **(50 puntos)**:

   a) el algoritmo elegido comentado;

   Solución:

   El algoritmo de unificación elegido fue el visto durante las lecturas de clase [1]:

```
function Unifica(E)
    repeat
        (s = t) ← seleccionar(E)
        % Primero se verifica que el functor de ambos terminos sea el mismo y
        ↪ que tengan aridad mayor o igual a cero
        if f(s1, ..., sn) = f(t1, ..., tn) (n ≥ 0) then
            % Si es el caso entonces se compara subtérmino a subtérmino
            remplazar (s = t) por s1 = t1, ..., sn = tn
        % En caso de no ser el mismo la unificación falla
        else if f(s1, ..., sm) = g(t1, ..., tn) (f/m != g/n) then
            return(fallo)
        % Si ambos términos son variables y además son la misma variable
        ↪ entonces se puede quitar del resto del proceso de unificación
        else if X = X then
            remover la X = X
        % Si es el caso en que solo uno de los terminos es una variable, el
        ↪ del lado derecho, entonces se cambia de orden
        else if t = X then
            remplazar t = X por X = t
        % Si es el caso en que el termino de la izquierda es una variable y el
        ↪ de la derecha no es una variable se debe realizar el chequeo de
        ↪ que la variable no sea subtérmino dentro del otro término
        else if X = t then
            % Si la variable aparece como subtérmino entonces la unificación
            ↪ falla
            if subtermino(X, t) then
                return(fallo)
            % Si no es subtérmino entonces se remplaza toda ocurrencia de X
            ↪ por el otro término
            else remplazar todo X por t
            end if
        end if
```

```
    % Se repiten las acciones anteriormente mencionadas hasta que ya no se
    ↪ pueda realizar más acción alguna
    until no hay acción posible para E
end function
```

---

b) su código, también comentado;

Solución:

```
%%% Autor: Leonardo Flores Torres
%%% Curso: Programacion para la Inteligencia Artificial
%%% Profesor: Alejandro Guerra Hernández
%%% Implementación del algoritmo de unificación

% Caso inicial: comienza el proceso de unificación comparando la cabeza de las
↪ metas, sus subtérminos y su aridad
unify(T1, T2) :-
    compound(T1),
    compound(T2),
    T1=..[F|Args1],
    T2=..[F|Args2],
    length(Args1, Len),
    length(Args2, Len),
    maplist(unify, Args1, Args2).

% Caso: T1 es una variable y T2 un termino compuesto, se hace chequeo de
↪ ocurrencias para verificar que T1 no sea un subtermino en T2.
unify(T1, T2) :-
    var(T1),
    compound(T2),
    unify_with_occurs_check(T1, T2).

% Caso: T1 es un atomo y T2 es una variable
unify(T1, T2) :-
    atomic(T1),
    var(T2),
    T2 = T1.

% Caso: T1 es una variable y T2 es un atomo
unify(T1, T2) :-
    var(T1),
    atomic(T2),
    T1 = T2.

% Caso: T1 es una variable y T2 tambien pero son diferentes
unify(T1, T2) :-
    var(T1),
    var(T2),
    T1 = T2.

```

```
40  % Caso: T1 y T2 son el mismo y ademas son variables
41  unify(T, T) :- var(T).
```

Se tomó como referencia el material de clase [1], y lo discutido durante las lecturas de la misma materia. También se revisó material incluido como respuesta en una página web [2]. El primer caso se muestra en la línea 6, aquí es donde primero se unifican las cabezas de las metas, en caso de ser iguales y de tener la misma aridad se procede con el algoritmo de unificación. Si es el caso en que se se procede con el algoritmo entonces se realiza un mapeo para aplicar el algoritmo de unificación ahora a todos los subtérminos por pares.

La línea 16 cubre el punto cuando es el caso que hay una variable y un término compuesto, es aquí que se hace el chequeo de ocurrencias. El caso de la linea 22 es aquel donde un término es una constante y el otro término es una variable, más específicamente en el orden $t = X$, lo que se hace es invertir la relación a $X = t$. De manera similar, en la línea 28 se maneja el caso entre variable y constante cuando $X = t$. Finalmente, los útimos dos casos a partir de la línea 34 son cuando ambos términos son variables, cuando son diferentes se toma $X = Y$ y se pueden remplazar las ocurrencias de Y por X, y cuando son iguales se puede remover la variable del resto del proceso de unificación.

---

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).

Solución:

Los ejemplos de ejecución del algoritmo de unificación se han puesto juntos y se muestran a continuación:

```
1  ?- unify(q(Y, g(a, b)), p(g(X, X), Y)).
2  false.
3
4  ?- unify(r(a, b, c), r(X, Y, Z)).
5  X = a,
6  Y = b,
7  Z = c .
8
9  ?- unify(mayor(padre(Y), Y), mayor(padre(Z), juan)).
10 Y = Z, Z = juan .
11
12 ?- unify(conoce(padre(X), X), conoce(W, W)).
13 false.
```

En el primer caso falla porque las cabezas de ambas metas son distintas, `q` y `p`; el segundo caso unifica la cabeza de ambas metas, y además unifican término por término, como todos los términos son constantes de un lado y variables del otro unifican rápidamente y se hace el cambio de `t = X` por `X = t`; en el cuarto es donde sucede el chequeo de ocurrencias y falla.

---

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.
```

Solución:

La regla que computa el valor de verdad respecto a si un conjunto es subconjunto de otro, `subset1/2`, se muestra a continuación:

```
%%% subset/2 returns the truth value if a given set is subset of another set, both sets represented as lists
%%% Examples:
% ?- subset1([1,3], [1,2,3,4]).
% true.
% ?- subset1([], [1,2]).
% true.
subset1([], _).
subset1([X|Xs], Y) :-
    member(X, Y), !,
    subset1(Xs, Y).

%%% subset2/2 makes use of subset/2 already predefined inside Prolog
subset2(X, Y) :-
    subset(X, Y).
```

Además, se incluyó una regla extra, `subset2/2`, que hace uso de la función ya definida en Prolog. En los comentarios de `subset1/2` se muestra la solución a la meta requerida para este inciso.

   - Intersección:

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

Solución:

La implementación de la regla que computa la intersección entre dos conjuntos representados como listas, `intersection1/2`, se muestra a continuación:

```
%%% intersection1/3 computes the intersection operation between two sets
↪  represented as lists
% ?- intersection1([1,2,3], [2,3,4], L).
% L = [2, 3].
intersection1(X, Y, Z) :-
    intersection1aid(X, Y, Z), !.
% base case
intersection1aid(_, [], []).
% case when Y is a member of Y
intersection1aid(X, [Y|Ys], [Y|Zs]) :-
    member(Y, X),
    intersection1aid(X, Ys, Zs).
% case when Y is not a member of X
intersection1aid(X, [_|Ys], Zs) :-
    intersection1aid(X, Ys, Zs).

%%% intersection2/3 makes use of intersection/3 alaready predefined inside
↪  prolog
% ?- intersection2([1,2,3], [2,3,4], L).
% L = [2, 3].
intersection2(X, Y, Z) :-
    intersection(X, Y, Z).
```

Se excribió una regla desde primeros principios que hiciera la intersección, `intersection1`, y también se incluyó otra regla, `intersection2/2`, que hiciera uso de la ya definida en la librería de Prolog.

- Unión:

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

Solución:

Para este inciso se definieron dos reglas principales, la primera es una implementación de la operación unión entre subconjuntos y la segunda es una regla usando la definición incluída en la librería de Prolog:

```
%%% union1/3 computes the union operation between two sets and returns
%%% the same elements as union/3 predefined in prolog
%%% Example:
% ?- union1([1,2,3,4], [2,3,4,5], L).
% L = [1, 2, 3, 4, 5] .
union1(X, Y, Z) :-
    union1aid(X, Y, Z1), !,
    sort(Z1, Z).
union1aid(X, [], X).
```

```
union1aid(X, [Y|Ys], [Y|Xs]) :-
    not(member(Y, X)),
    union1aid(X, Ys, Xs).
union1aid(X, [Y|Ys], Xs) :-
    member(Y, X),
    union1aid(X, Ys, Xs).

%%% union2/3 makes use of union/3 already predefined inside prolog
%%% Example:
% ?- union2([1,2,3,4], [2,3,4,5], L).
% L = [1, 2, 3, 4, 5].
union2(X, Y, Z) :-
    union(X, Y, Z).
```

Uno puede corroborar que si se borra el `sort/2` de `union1/3` el resultado es el mismo que el mostrado en el ejemplo, solo con la diferencia de que los elementos no están ordenados. De hecho, el ejemplo que se menciona en el comentario de ambas reglas es el que se menciona para corroborar este inciso.

- Diferencia:

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

Solución:

La regla definida para calcular la diferencia entre dos conjuntos se muestra a continuación:

```
%%% difference/3 computes the difference between two given sets represented as
↪  lists
difference1(X, Y, Z) :-
    difference1aid(X, Y, Z), !.
difference1aid([], _, []).
difference1aid([X|Xs], Y, Z) :-
    member(X, Y),
    difference1aid(Xs, Y, Z).
difference1aid([X|Xs], Y, [X|Zs]) :-
    difference1(Xs, Y, Zs).
```

Probando las metas con los conjuntos mencionados para este inciso se obtiene la respuesta esperada:

```
?- difference1([1,2,3], [1,4,5], L).
L = [2, 3].
?- difference1([1,2,3,4], [2,3,4,5], L).
L = [1].
```

Quisiera mencionar que para algunas de estas soluciones tomé como guía una preguntas en la red sobre como unir listas en prolog [3] y otra sobre como encontrar la intersección entre dos listas [4].

---

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:

Los números de Peano son una representación de los números naturales, y como tal pueden tomar valores negaticos, esto se tomó en cuenta para la implementación de las reglas de este inciso. Primero definí una manera de convertir números naturales en la representación de Peano:

```
%%% Withouth finite domains
%%% natToPeano/2 computes the conversion of natural numbers to their Peano
↪  representation, it also deals with negative natural numbers
%%% Examples:
% ?- natToPeano(3, X).
% X = s(s(s(0))) .
% ?- natToPeano(-3, X).
% X = -s(s(s(0))) .
natToPeano(0, 0).
natToPeano(N, s(X)) :-
    N >= 0,
    N1 is N - 1,
    natToPeano(N1, X).
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, it also deals with negative Peano numbers
%%% Examples:
% ?- peanoToNat(s(s(s(0))), X).
% X = 3.
% ?- peanoToNat(-s(s(s(0))), X).
% X = - 3.
peanoToNat(0, 0).
peanoToNat(s(X), N) :-
    peanoToNat(X, N1),
    N is N1 + 1.
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, it can deal with positive and negative Peano numbers alike
%%% Examples:
% ?- addPeano(s(0), 0, X).
% X = s(0) .
% ?- addPeano(s(0), s(s(0)), X).
% X = s(s(s(0))) .
% ?- addPeano(-s(0), s(s(0)), X).
% X = s(0) .
% ?- addPeano(s(0), -s(s(0)), X).
% X = -s(0) .
addPeano(S1, S2, S3) :-
```

```
    peanoToNat(S1, N1),
    peanoToNat(S2, N2),
    N3 is N1 + N2,
    natToPeano(N3, S3).
```

De manera similar, la operación de resta entre dos números de Peano se muestra a continuación:

```
%%% substractPeano/3 substracts S2 to S1, S3 = S1 - S2; it can deal with
↪  operations between positive and negative numbers alike
%%% Examples:
% ?- substractPeano(s(0), 0, X).
% X = s(0) .
% ?- substractPeano(0, s(0), X).
% X = -s(0) .
% ?- substractPeano(s(0), s(s(0)), X).
% X = -s(0) .
% ?- substractPeano(s(0), -s(s(0)), X).
% X = s(s(s(0))) .
substractPeano(S1, S2, S3) :-
    peanoToNat(S1, N1),
    peanoToNat(S2, N2),
    N3 is N1 - N2,
    natToPeano(N3, S3).
```

Para ambas operaciones consideré oportuno hacer el cambio de ambos números de peano a su respectivo número natural, realizar la operación correspondiente y regresar a la representación de Peano. La operación resta entre números de peano se consideró como una operación en la que el orden importa, esto es, $p_1 - p_2$ donde p_1 es el número al que se le resta, y ambos números de Peano pueden ser tanto positivos como negativos. Además, si no quisiera tomarse en consideración el órden de la operación como en `substractPeano/2` podría tomarse la regla `addPeano/2` para asignarle un signo negativo al número de Peano deseado y así tener un equivalente de la resta. No se incluyeron casos de prueba como para otros incisos ya que se muestran como comentarios de sus respectivas reglas. Para responder a este inciso se tomó como referencia una pregunta de la web [5] y un repositorio de GitHub [6].

---

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 inclinado:

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

```
 *     *     *     *     *     *
*     *     *     *     *     *     *
true.
```

Tomé como referencias de apoyo unas preguntas en la red, [7] y [8], para guiarme al responder este inciso.

---

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. Para resolver este inciso tomé como referencias algunas preguntas en la web, [9] y [10].

Referencias

[1] Alejandro Guerra-Hernandez. Programación para la inteligencia artificial. https://www.uv.mx/personal/aguerra/pia/, 2022. Visitado: 2022-10-07.

[2] StackOverflow. The unification algorithm in prolog. https://stackoverflow.com/questions/64638801/the-unification-algorithm-in-prolog, 2021. Visitado: 2022-10-24.

[3] StackOverflow. Making a union of two lists in prolog. https://stackoverflow.com/questions/46899153/making-a-union-of-two-lists-in-prolog, 2017. Visitado: 2022-10-25.

[4] StackOverflow. Intersection and union of 2 lists. https://stackoverflow.com/questions/9615002/intersection-and-union-of-2-lists, 2012. Visitado: 2022-10-23.

[5] StackOverflow. Convert peano number s(n) to integer in prolog. https://stackoverflow.com/questions/8954435/convert-peano-number-sn-to-integer-in-prolog, 2012. Visitado: 2022-10-26.

[6] Fredrik Fagerholm. Github repository: ffagerholm/peano. https://github.com/ffagerholm/peano, 2016. Visitado: 2022-10-26.

[7] StackOverflow. How can i draw star triangle using recursive in prolog? https://stackoverflow.com/questions/20009868/how-can-i-draw-star-triangle-using-recursive-in-prolog, 2014. Visitado: 2022-10-23.

[8] StackOverflow. How to draw right triangle using recursion in prolog? https://stackoverflow.com/questions/65277514/how-to-draw-right-triangle-using-recursion-in-prolog, 2021. Visitado: 2022-10-23.

[9] StackOverflow. gnu prolog powerset modification. https://stackoverflow.com/questions/4146117/gnu-prolog-powerset-modification, 2011. Visitado: 2022-10-25.

[10] StackOverflow. Prolog compute the permutation. https://stackoverflow.com/questions/12824685/prolog-compute-the-permutation, 2012. Visitado: 2022-10-25.