Tarea Independiente 16/10/2025

David Núñez Franco

October 20, 2025

Inventario de Conceptos Claves

- Uso básico de swipl para ejecutar scripts
- Prolog no tiene while, for, if-then-else explícitos
- Prolog: diseñado para recursión + backtracking
- Prolog como probador de teoremas en lógica de Horn + Lógica del Usuario (Business Logic)
- Probar == Computar respuestas (true) o falla (false, fail)
- Razona por búsqueda exhaustiva "hacia-atrás" con backtracking (DFS, brute-force)
- Facts, Clauslas y Goals. Coma es AND, Punto-y-coma es OR, + es negación (por falla)
- Términos como AST
- Sin estructura:
  - Variables
  - Constantes o atómicos; átomos (comillas simples), números (lo usual), strings(comillas dobles)
  - Ejemplos: juan, '100-ABC', "Hello World!"
- Compuestos (términos): listas ([], [1, juan, [2] [[]]], tuple(juan, 20)
  - Ejemplos person(juan, 20, male) accounts('100-ABC', 10000)
- Un término tiene semántica solo si tiene reglas asociadas o está definido en Prolog
- Programas como Pipelines de datos+control o dataflows
- Computación por backtracking (equivale a ciclos en la VM)
- Búsqueda de "pruebas" DFS

Ejercicio 1

Escriba accounts_print(+P) que imprime las cuentas de la persona P una por una c/ líne
a) Usando solo member, writeln y false. Nota: false es un predicado que siempre provoc
b) Usando el predicado forall/2 (ver en documentación)

Solución

```
/*Ejercicio 01 tarea independiente 16-10-2025*/

person_account(juan, '200-ABC').
person_account(juan, '200-QWR').
person_account(juan, '100-ABC').
person_account(maria, '200-ABC').
person_account(maria, '100-RST').

/*
select columns from table where filter (condition)
P = Persona
Accs = Lista de cuentas
+ entran datos, - salen datos, ? significa entrada/salida
*/
%- accounts_of(+P, -Accs)
accounts_of(P, Accs) :-
findall(Acc, person_account(P, Acc), Accs).

% a) member + writeln + false (fail-driven loop)
accounts_print_a(P) :-
accounts_of(P, Accs), % busca todas las cuentas de la
   persona P y las guarda en la lista Accs
member(Acc, Accs), % toma una cuenta (Acc) de la lista, una
   por una
writeln(Acc), % imprime esa cuenta
false; % provoca backtracking: obliga a Prolog a volver a
   member y tomar otra cuenta
true. % cuando ya no queden mas cuentas, termina
   correctamente

% b) usando forall
accounts_print_b(P) :-
accounts_of(P, Accs),
forall(member(Acc, Accs), writeln(Acc)). % Para cada Acc en
   Accs, ejecuta writeln(Acc).
```

Listing 1: Ejercicio 1.a y 1.b

```
?- accounts_print_a(juan).
200-ABC
200-QWR
100-ABC
true.

?- accounts_print_b(maria).
200-ABC
100-RST
true.
```

Listing 2: output

Ejercicio 2

Haga Holamundo.pl version nuev:
Cambie lo representación de persona como átomo por una estructura. Use un término así:
Por ejemplo
person(name(juan, perez), 20, male) % persona nombre de pila juan, apellido perez, eda

b) Añada getters (para nombre (de pila o apellido), edad y género usando facts. Por Ej
person_name(person(Name,_,_), Name).
% Otros getters para age y gender.

b) Escriba un predicado account_invalid_age(-InvAccs) que calcule en InvAccs una lista

Solución

```
/*
@author: david
holamundo.pl (version nueva con person(name(First,Last),
   Gender, Age))
*/

% Personas + Cuentas (antes atomos, ahora terminos)
person_account(person(name(juan, perez), male, 20), '200-ABC
   ').
person_account(person(name(juan, perez), male, 20), '200-QWR
   ').
person_account(person(name(juan, perez), male, 20), '100-ABC
   ').
person_account(person(name(maria, lopez), female, 16), '200-
   ABC').
```

```
person_account(person(name(maria, lopez), female, 16), '100-RST').

% Getters (facts, reglas puras)
person_name(person(Name, _, _), Name).
person_first_name(person(name(First, _), _, _), First).
person_last_name(person(name(_, Last), _, _), Last).
person_age(person(_, _, Age), Age).
person_gender(person(_, Gender, _), Gender).

/*
select columns from table where filter (condition)
P = Persona (term person/3)
Accs = Lista de cuentas
+ entran datos, - salen datos, ? entrada/salida
*/
%- accounts_of(+P, -Accs)
accounts_of(P, Accs) :-
findall(Acc, person_account(P, Acc), Accs).

/*
account_invalid_age(-InvAccs)
Construye todos los invalidos por minoria de edad y los ordena por apellido.
Forma: invalid(name(Last, First), Age, Acc)
*/
account_invalid_age(InvAccs) :-
% genera una lista raw con todas las cuentas invalidas
findall(
invalid(name(Last, First), Age, Acc), % Estructura de cada elemento
( person_account(P, Acc),             % Busca persona y cuenta
person_age(P, Age),                   % Obtiene edad
Age < 18,                             % Filtro: menor de edad
person_first_name(P, First),          % Obtiene nombre de pila
person_last_name(P, Last)             % Obtiene apellido
),
Raw
),
sort(Raw, InvAccs). % Ordena alfabeticamente y elimina duplicados
```

Listing 3: holamundo.pl

```
?- [holamundo].
true.

```

```
?- person_name(person(name(juan,perez), male, 20), N).
N = name(juan, perez).

?- person_age(person(name(maria,lopez), female, 16), A).
A = 16.

?- accounts_of(person(name(juan,perez), male, 20), L).
L = ['200-ABC', '200-QWR', '100-ABC'].

?- account_invalid_age(X).
X = [invalid(name(lopez, maria), 16, '100-RST'), invalid(
   name(lopez, maria), 16, '200-ABC')].
```

Listing 4: output

Uso de predicado !

```
/*
Ejemplo de uso del predicado !

cut (!) detiene el backtracking.
Es decir, una vez que Prolog llega al corte,
no volvera atras para probar otras opciones.

Predicado: grade(+Score, -Result)
Asigna una calificacion (Result) segun la nota (Score).
*/

% Caso 1: si la nota es 90 o mas, es excelente.
grade(Score, excellent) :-
    Score >= 90, !.        % Corte: no se evaluan reglas
       siguientes.

% Caso 2: si la nota es 70 o mas, es aprobada.
grade(Score, pass) :-
    Score >= 70, !.        % Otro corte para detener al
       cumplir esta condicion.

% Caso 3: en cualquier otro caso, es reprobada.
grade(_, fail).

/*
Como funciona paso a paso:

?- grade(95, R).
1) Coincide con la primera regla (Score >= 90).
```

```
        2) Llega al ! -> corta las demas opciones.
        3) Resultado: R = excellent.

        ?- grade(75, R).
        1) Falla la primera (75 >= 90 -> falso).
        2) Pasa a la segunda (75 >= 70 -> verdadero).
        3) Llega al ! -> detiene busqueda.
        4) Resultado: R = pass.

        ?- grade(40, R).
        1) Falla las dos primeras.
        2) Solo queda la tercera.
        3) Resultado: R = fail.
        */
```

```
        ?- [ejercicioCorte].
        true.

        ?- grade(95, R).
        R = excellent.

        ?- grade(75, R).
        R = pass.

        ?- grade(40, R).
        R = fail.
```

Listing 5: output