

## Programación Lógica - Parte 2

### Paradigmas (de Lenguajes) de Programación

Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

8 de noviembre de 2024

Por convención se aclara mediante prefijos en los comentarios:

- $p(+A)$  indica que  $A$  debe proveerse instanciado.
- $p(-A)$  indica que  $A$  no debe estar instanciado.
- $p(?A)$  indica que  $A$  puede o no proveerse instanciado.
- Existe un último caso en donde un argumento puede aparecer **semi instanciado** (es decir, contiene variables libres), por ejemplo:  
 $[p,r,o,X,o,-]$  unifica con  $[p,r,o,l,o,g]$  pero no con  $[]$  o  $prolog$ .

#### Predicados útiles

- $var(A)$  tiene éxito si  $A$  **es** una variable libre.
- $nonvar(A)$  tiene éxito si  $A$  **no** es una variable libre.
- $ground(A)$  tiene éxito si  $A$  **no contiene** variables libres.

## Ejercicio

### Ejercicio: iésimo

- Implementar el predicado  $iesimo(+I, +L, -X)$ , donde  $X$  es el iésimo elemento de la lista  $L$ .
- ¿Es nuestra implementación reversible en  $I$ ? Si no lo es, hacer una versión reversible.

## Ejercicio

### El predicado desde.

$desde(X, X).$

$desde(X, Y) :- N \text{ is } X+1, desde(N, Y).$

### Ejercicio: desde

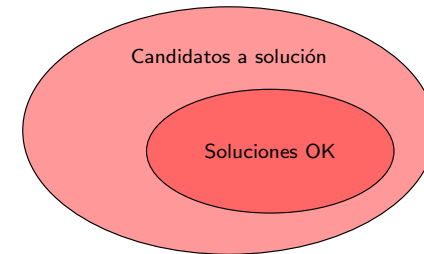
- ¿Cómo deben instanciarse los parámetros para que el predicado funcione? (es decir, para que no se cuelgue ni produzca un error). ¿Por qué?
- Implementar el predicado  $desdeReversible(+X, ?Y)$  tal que si  $Y$  está instanciada, sea verdadero si  $Y$  es mayor o igual que  $X$ , y si no lo está, genere todos los  $Y$  de  $X$  en adelante.

Definir el predicado `pmq(+X, -Y)` que genera todos los naturales pares menores o iguales a `X`.

Una técnica que usaremos muy a menudo es:

- 1 Generar todas las posibles soluciones de un problema.  
(Léase, los *candidatos* a solución, según cierto criterio general)
- 2 Testear cada una de las soluciones generadas.  
(Hacer que fallen los candidatos que no cumplan cierto criterio particular)

La idea se basa fuertemente en el *orden* en que se procesan las reglas.



Un predicado que usa el esquema G&T se define mediante otros dos:

```
pred(X1,...,Xn) :- generate(X1, ...,Xm), test(X1, ...,Xm).
```

Esta división de tareas implica que:

- `generate(...)` deberá **instanciar** ciertas variables.
- `test(...)` deberá **verificar** si los valores instanciados pertenecen a la solución, pudiendo para ello asumir que ya está instanciada.

### Ejercicio

- Definir el predicado `coprimos(-X, -Y)` que instancia en `X` e `Y` **todos** los pares de números coprimos. (Tip: utilizar la función `gcd` del motor aritmético: `X is gcd(2, 4)` instancia `X=2`).
- ¿Es reversible en `X` e `Y`? Justificar.

## Soluciones repetidas

### Algunos hechos sobre materias de cierta carrera

```
altaMateria(plp).
altaMateria(aa).
altaMateria(metnum).

liviana(plp).
liviana(aa).
liviana(eci).

obligatoria(plp).
obligatoria(metnum).

leGusta(M) :- altaMateria(M).
leGusta(M) :- liviana(M).

hacer(M) :- leGusta(M), obligatoria(M).
```

### Consulta

```
?- hacer(Materia).
```

### Resultados

```
Materia = plp ;
Materia = metnum ;
Materia = plp ;
false.
```

- ¿Razonable o erróneo?
- ¿Cómo hacer para evitar repeticiones no deseadas?

## Cómo evitar soluciones repetidas

Idea 1: Usando el **metapredicado** `setof` y `member`

### setof

`setof(-Var, +Goal, -Set)`

unifica Set con la lista *sin repetidos* de Var que satisfacen Goal.

### Uso

- `setof(X, p(X), L)` instancia L en el conjunto de X tales que p(X).
- Un ejemplo:  

```
primeraComponente([(X,_)|_],X).
primeraComponente([_|XS],X) :- primeraComponente(XS,X).

?- setof(X,primeraComponente([(2,2),(1,3),(1,4)],X),L).
L = [1,2].
```

Utilizando `setof` hacer otra versión del predicado `hacer(M)` en donde no haya soluciones repetidas.

## El metapredicado not

### Definición

```
not(P) :- call(P), !, fail.
not(P).
```

- `not(p(X1, ..., Xn))` tiene éxito si **no existe** instanciación posible para las **variables no instanciadas** en  $\{X1 \dots Xn\}$  que haga que P tenga éxito.
- el `not` **no deja instanciadas** las variables libres luego de su ejecución.

## Cómo evitar soluciones repetidas

Idea 2: Usando cláusulas excluyentes.

### Algunos hechos sobre materias de cierta carrera

```
altaMateria(plp).
altaMateria(aa).
altaMateria(metnum).

liviana(plp).
liviana(aa).
liviana(eci).

obligatoria(plp).
obligatoria(metnum).

leGusta(M) :- altaMateria(M).
leGusta(M) :- liviana(M), not(altaMateria(M)).

hacer(M) :- leGusta(M), obligatoria(M).
```

### ¡Esto no funciona! ¿Por qué?

```
leGusta(M) :- altaMateria(M).
leGusta(M) :- not(altaMateria(M)), liviana(M).
```

## Ejercicio

Definir el predicado `corteMásParejo(+L,-L1,-L2)` donde `L` es una lista de números, y `L1` y `L2` representan el corte más parejo posible de `L` respecto a la suma de sus elementos (predicado `sumlist/2`). Puede haber más de un resultado.

```
corteMásParejo([1,2,3,4,2],I,D). ~> I = [1, 2, 3],
                                   D = [4, 2] ;
                                   false.
corteMásParejo([1,2,1],I,D).      ~> I = [1], D = [2, 1] ;
                                   I = [1, 2], D = [1] ;
                                   false.
```

## Ejercicio

Definir el predicado `próximoPrimo(+N,-P)` que instancia en `P` el siguiente número primo a partir de `N`.

```
próximoPrimo(32,P). ~> P = 37;
                    false.
próximoPrimo(37,P). ~> P = 41;
                    false.
```

## Generación infinita: triángulos

Suponiendo que los triángulos se representan con `tri(A,B,C)` cuyos lados tienen longitudes `A`, `B` y `C` respectivamente. Se asume que las longitudes de los lados son siempre números naturales mayores a cero. Se cuenta con el predicado `esTriangulo(+T)` que es verdadero cuando `T` es una estructura de la forma `tri(A,B,C)` que representa un triángulo válido (cada lado es menor que la suma de los otros dos, y mayor que su diferencia).

## Ejercicio

- Implementar un predicado `perímetro(?T,?P)` que es verdadero cuando `T` es un triángulo y `P` es su perímetro. No se deben generar resultados repetidos (no tendremos en cuenta la congruencia entre triángulos: si dos triángulos tienen las mismas longitudes, pero en diferente orden, se considerarán diferentes entre sí). El predicado debe funcionar para cualquier instancia de `T` y `P` (no es necesario que funcione para triángulos parcialmente instanciados).
- Implementar un generador de triángulos válidos, sin repetir resultados: `triángulo(-T)`.