Lenguajes de programación

Programando en PROLOG

Ejecución de Programas y Programación Recursiva

¿Cómo funciona PROLOG?

- En Prolog NO existen instrucciones de control.
- Su ejecución se basa en dos conceptos:
  - la unificación y
  - el backtracking.
- Gracias a la unificación, cada meta determina un subconjunto de cláusulas susceptibles de ser ejecutadas.
- Cada una de ellas se denomina *punto de elección*.

Backtracking

- Prolog selecciona el primer punto de elección y sigue ejecutando el programa hasta determinar si la meta es verdadera o falsa.
- En caso de que un punto de elección sea falso entra en juego el **backtracking**.
- El backtracking consiste en deshacer todo lo ejecutado situando el programa en el mismo estado en el que estaba justo antes de llegar al punto de elección.
- Entonces se toma el siguiente punto de elección que estaba pendiente y se repite de nuevo el proceso.

¿Cómo trabaja PROLOG?

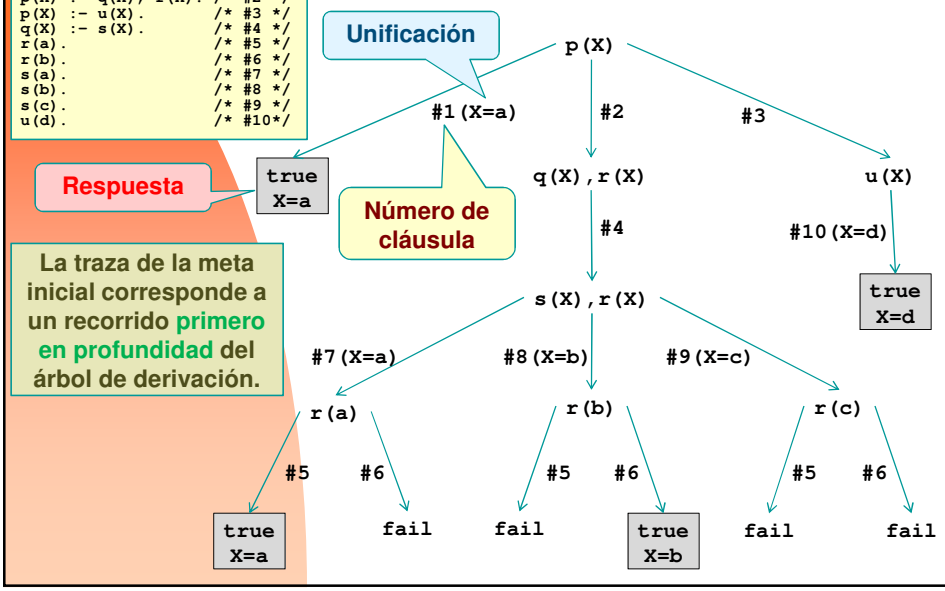
Para ilustrar como produce Prolog las respuestas para programas y metas, considera el siguiente programa.

```
/* programa P */
p(a).                  /* #1 */
p(X) :- q(X), r(X).    /* #2 */
p(X) :- u(X).          /* #3 */
q(X) :- s(X).          /* #4 */
r(a).                  /* #5 */
r(b).                  /* #6 */
s(a).                  /* #7 */
s(b).                  /* #8 */
s(c).                  /* #9 */
u(d).                  /* #10*/
```

- **Ejercicio1**:
  1. Carga el programa P en Prolog
  2. Observa lo que pasa para la meta: ?- p(X).
  3. Usar ; para desplegar todas las respuestas.
- **Ejercicio2**:
  1. Carga el programa P en Prolog
  2. Activa la traza con trace.
  3. Observa lo que pasa para la meta: ?- p(X).
  4. Usar ; para desplegar todas las respuestas.

Ejecución para p(X).

```
/* programa P */
p(a).                 /* #1 */
p(X) :- q(X), r(X).   /* #2 */
p(X) :- u(X).         /* #3 */
q(X) :- s(X).         /* #4 */
r(a).                 /* #5 */
r(b).                 /* #6 */
s(a).                 /* #7 */
s(b).                 /* #8 */
s(c).                 /* #9 */
u(d).                 /* #10*/
```



Programación en Prolog

- Operadores RELACIONALES:
  - igual que **=:=**
  - diferente **=\=**
  - menor que **<**
  - mayor que **>**
  - menor igual que **=<**
  - mayor igual que **>=**
- Conforman cláusulas en formato infijo y por si mismos arrojan un valor de verdad.

Ejemplo

```
goberno(vfq, 2000, 2006)
goberno(ezp, 1994, 2000).
goberno(csg, 1988, 1994).
goberno(mmh, 1982, 1988).
goberno(jlp, 1976, 1982).
goberno(lea, 1970, 1976).

fue_presidente(Persona, Anio):-
  goberno(Persona, Inicio, Fin),
  Anio >= Inicio, Anio =< Fin.
```

Programación en Prolog

- Operadores ARITMÉTICOS:
  - suma **+**
  - resta **-**
  - multiplicación *
  - división /
  - residuo **mod**
- Conforman cláusulas en formato infijo cuyo resultado debe instanciarse a una variable a través del operador **is**.

Operador is

- Es requisito utilizarlo cuando se requiere hacer una evaluación aritmética.
- Formato:

  *Variable **is** expresión_aritmética*
- Instancía a la variable con el resultado de la expresión, y la cláusula es **VERDADERA** por default.

Ejemplo

- Suponer que se tienen definidos HECHOS para:

```
poblacion(pais, cantidad).
area(pais, espacio).
```

```
densidad(Pais, D) :-
    poblacion(Pais, P),
    area(Pais, A),
    D is P/A.
```

Reglas como módulos

- Bajo un enfoque de la abstracción modular, las reglas son MÓDULOS, y las variables de la regla, son parámetros de entrada y/o salida según el caso.

Entrada *Salida*

```
densidad(Pais, D) :-
  poblacion(Pais, P),
  area(Pais, A), D is P/A.
```

Mecanismos de Control

- **No existe la iteración!**
- …aunque se puede simular algo parecido así:

```
prueba :-
  goberno(P,_,_),
  write(P),nl,
  fail.
prueba.

?- prueba.
  vfq
  ezp
  …
```

Generador de soluciones

Cuerpo del ciclo

test (en este caso, siempre provoca backtracking!)

Alternativa: reglas recursivas

- Reglas cuyo cuerpo, tienen términos que corresponden a la cabeza de la propia regla.
- Ejemplo:

```
abuelo(X,Y) :- padre(X, Z), padre(Z,Y).
bisabuelo(X,Y) :- padre(X, Z),
                  abuelo(Z,Y).
tatarabuelo(X,Y):- padre(X, Z),
                   bisabuelo(Z,Y).
…
ancestro(X,Y) :- padre(X,Y).
ancestro(X,Y) :- padre(X,Z),
                 ancestro(Z, Y).
```

Pensamiento recursivo

- Se aplica de la misma manera.
- La implementación involucra tener al menos una regla para el caso base, y al menos una regla recursiva (que se llama a sí misma).
- La decisión de evaluar un caso o el otro, está implícita en la manera en que trabaja el intérprete.

Ejemplo

- Factorial de un número.
- Relación entre un número y su factorial.
- CASO BASE: **0! = 1**

```
factorial(0, 1).
```

- CASO GENERAL: **n! = n * (n-1)!**

```
factorial(N, R) :-
        X is N-1,
        factorial(X, W),
        R is N*W.
```

Errores comunes

$$x * y = \begin{cases} 0 & \text{si } x = 0 \\ (x-1) * y + y & \text{si } x > 0 \end{cases}$$

```
producto(0,X,0).
producto(X,Y,W+Y) :- X>0, producto(X-1,Y,W).

producto(0,X,0).
producto(X,Y,Z) :- X>0, producto(X-1,Y,W), Z is W+Y.

producto(0,X,0).
producto(X,Y,Z) :- X>0, X is X-1, producto(X,Y,W), Z is W+Y.
```

anidar operaciones

anidar operaciones

asignación destructiva

Solución correcta:

```
producto(0,X,0).
producto(X,Y,Z) :- X>0, N is X-1, producto(N,Y,W), Z is W+Y.
```