Prolog

Entradas/Salidas
Operaciones aritméticas
Negación, Disyunción
Cortes

Entradas y Salidas

- La cláusula write/1 permite escribir en pantalla.
- Además, las cláusulas nl/0 y tab/1 permiten generar un avance de línea y tabulaciones de n espacios respectivamente.

```
holaMundo :- tab(5), write('Hola Mundo'), nl.
```

- Una cláusula sin encabezado es ejecutada al consultar el programa:

```
:- write('Bienvenido al programa'), nl.
```

Entradas y salidas

- La cláusula read/1 lee datos de consola y los unifica con la variable

```
nombre(X) :- write('Ingrese un nombre: '), read(X), nl.
tocayos(X,Y) :- nombre(X), nombre(Y), X=Y,
                write('X y Y son tocayos').
```

Operadores Aritméticos Básicos

Precedencia	Operador	Significado
1	+, -	Signo
2	^, **	Potenciación
3	*, /, mod, //	Multiplicación, división, modulo, división entera
4	+, -	Adición, sustracción
5	<, >, =:=, =\=, =<, >=, is	Operadores relacionales, evaluación

Operadores de un mismo nivel de precedencia se evalúan de izquierda a derecha.

El operador *is*

- El predicado '*is*' evalua la expresión aritmética del lado derecho y la unifica con el lado izquierdo. Por ejemplo

```
?- Y is 2+7.
Y = 9.
```

- Notar que la evaluación es del lado derecho. El siguiente ejemplo ilustra usos correctos e incorrectos de '*is*'

```
?- 7 is 2+Y.
ERROR: is/2: Arguments are not sufficiently instantiated

?- 3+4 is 7.
false.

?- 7 is 3+4.
true.
```

Operadores relacionales

- Evalúan las expresiones a ambos lados y aplican el respectivo operador.
- Ejemplos:

```
?- 3+4 =:= 2+5.
true.

?- 2*5 =\= 9*3.
true.

?- 3<5, 4=<2*2, 7*2>=3^2, 9**2>8*10.
true.

?- 20 =:= (20//3)*3 + 20 mod 3.
true.
```

Funciones

- Algunas funciones matemáticas de uso frecuente:

Función	Descripción
abs/1	Valor absoluto
sqrt/1	Raíz cuadrada
sin/1, cos/1, tan/1	Seno, coseno, tangente
log/1, exp/1	Logaritmo y exponencial
min/2, max/2	Mínimo y máximo de 2 valores
float/1	Convertir a punto flotante
round/1	Redondear

Ejemplos funciones

```
?- PI is 3.14159, X is sin(PI/2), Y is cos(PI/3), Z is tan(PI/6).
PI = 3.14159,
X = 1.0,
Y = 0.500001,
Z = 0.57735.

?- X is abs(-1).
X = 1.

?- X is sqrt(-1).
ERROR: sqrt/1: Arithmetic: evaluation error: `undefined'
?- X is sqrt(27).
X = 5.19615.

?- X is min(10,20), Y is max(10,20).
X = 10,
Y = 20.

?- X is float(7), Y is round(5.59).
X = 7.0,
Y = 6.
```

Negación en Prolog

- Una cláusula de la forma:

$$\text{encabezado :- meta}_1,\ldots,\text{meta}_n$$

equivale a la expresión lógica

$$\text{meta}_1 \wedge \ldots \wedge \text{meta}_n \Rightarrow \text{encabezado}$$

Que reemplazando la implicación equivale a

$$\neg(\text{meta}_1 \wedge \ldots \wedge \text{meta}_n) \vee \text{encabezado}$$

$$\neg\text{meta}_1 \vee \ldots \vee \neg\text{meta}_n \vee \text{encabezado}$$

Todas las metas se niegan y solo el encabezado queda sin negar.

Cláusulas *Horn*

- Son cláusulas que con contienen como máximo un literal no negado
- Todas las reglas en Prolog son cláusulas *Horn*.
- Esto hace difícil expresar condiciones como por ejemplo:

  ¬ tieneTrabajo(X) ⇒ estaDesempleado(X)

  ~~estaDesempleado(X) :- ¬ tieneTrabajo(X).~~

Predicado “no es probable”

- Los hechos indicados en la base de datos son las únicas verdades que conoce el Prolog.
- Si un predicado no puede ser probado (no hay un hecho o regla que lo soporte), se asume que es falso.
- El predicado \+ devuelve verdadero si una meta no es probable.

```
tieneTrabajo(juan).
tieneTrabajo(maria).

estaDesempleado(X) :- \+ tieneTrabajo(X).
```

```
?- estaDesempleado(maria).
false.

?- estaDesempleado(pedro).
true.
```

La aplicabilidad \+ es limitada

- Consideremos el siguiente ejemplo

```
roja(rosa).
verde(hierba).
blanca(margarita).
```

```
?- \+ roja(X).
false.
```

- El predicado \+ no particulariza las variables.
- Si la variable no ha sido particularizada previamente, siempre falla.

Para \+ el orden importa

- Dado que la particularización ocurre en el orden que aparecen las metas, es distinto si el predicado \+ se evalúa antes o después de que ocurre la particularización de una variable:

Ejemplo:

```
carnivoro(león).
herbivoro(vaca).

vegetariano(X) :-
    \+ carnivoro(X), herbivoro(X).
```

```
?- vegetariano(X).
false.
```

Falla porque la X no se ha particularizado al momento de evaluar "`carnivoro`(X)"

Para \+ el orden importa

- La siguiente versión produce el resultado esperado

```
carnivoro(león).
herbivoro(vaca).

vegetariano(X) :-
    herbivoro(X), \+ carnivoro(X).
```

```
?- vegetariano(X).
X = vaca.
```

Disyunción lógica

- Prolog no tiene operador disyunción explícito.
- La disyunción se logra definiendo múltiples reglas, por ejemplo:

```
% signo/1 : Implementado como múltiples reglas
signo(X) :- X<0, write('Negativo').
signo(X) :- X>0, write('Positivo').
```

- SWI-Prolog permite utilizar ; para escribir múltiples reglas en una sola línea, por ejemplo:

```
% signo2/1 : Implementado en una sola línea
signo2(X) :- X<0, write('Negativo'); X>0, write('Positivo').
```

Disyunción

- Las reglas separadas por ; son independientes y el ámbito de las variables está retringido a su respectiva regla.

```
% ambito/0: Ilustra el ambito de las variables de cada regla
ambito :- X is pi, write('X='), write(X), nl, write('Y='), write(Y);
      Y is e, write('X='), write(X), nl, write('Y='), write(Y).
```

```
?- ambito.
X=3.141592653589793
Y=_L136
true ;
X=_L135
Y=2.718281828459045
true.
```

Evaluación “Corto circuito”

- Se optimiza la conjunción de muchas metas retornando inmediantamente cuando una de las metas es falsa.
  Ejemplo:

```
% shortCircuit/0: Ilustra la operación de la evaluación corto circuito
shortCircuit :- write('Parte A1'), pi<e, write('Parte A2').
```

```
?- shortCircuit.
Parte A1
false.
```

Cortes (cuts)

Consideremos el siguiente problema:

- Los impuestos a pagar se calculan así:
  - Si los ingresos son menores a 1000, se paga el 10%.
  - Si los ingresos están entre 1000 y 10000 se paga el 10% por los primeros 1000 y el 15% por el valor restante.
  - Si los ingresos son superiores a 10000 se paga el valor correspondiente a los primeros 10000, y el 20% sobre el valor restante.

Ejemplo: Cálculo del Impuesto

- Si el cálculo del impuesto se codifica así, ocurría que muchas reglas coinciden con la consulta:

```
valorImpuesto(Ingreso, Impuesto) :- Ingreso<1000,
                                    Impuesto is Ingreso*0.1.
valorImpuesto(Ingreso, Impuesto) :- Ingreso<10000,
                                    Impuesto is 100+(Ingreso-1000)*0.15.
valorImpuesto(Ingreso, Impuesto) :- Ingreso<10000,
                                    Impuesto is 1450+(Ingreso-10000)*0.2.
```

```
?- valorImpuesto(100, I).
I = 10.0 ;
I = -35.0 ;
I = -530.0.
```

Cortes (cuts)

- En realidad lo que queremos es que la primera regla que coincida se evalúe, pero no las siguientes.
- El símbolo corte ! fija las decisiones tomadas hasta el momento:
  - Variables previamente particularizadas
  - Metas ya satisfechas
  - La cláusula seleccionada
- No se hace retroceso (backtracking) más allá del corte, ni se evalúan otras cláusulas.

Cálculo del Impuesto - 2

- Haciendo un corte cuando se satisface la primera meta:

```
valorImpuesto(Ingreso, Impuesto) :- Ingreso<1000, !
                                    Impuesto is Ingreso*0.1.
valorImpuesto(Ingreso, Impuesto) :- Ingreso<10000, !
                                    Impuesto is 100+(Ingreso-1000)*0.15.
valorImpuesto(Ingreso, Impuesto) :- Ingreso<100000, !
                                    Impuesto is 1450+(Ingreso-10000)*0.2.
```

```
?- valorImpuesto(100, I).
I = 10.0

?-
```