

PROGRAMACIÓN DECLARATIVA

PROGRAMACIÓN LÓGICA

Tema PL2: El lenguaje PROLOG, aspectos básicos

5. Aritmética

Grado en Ingeniería Informática

URJC

Ana Pradera

Contenido

- 1 INTRODUCCIÓN
- 2 CONSTANTES Y OPERADORES ARITMÉTICOS
- 3 PREDICADOS ARITMÉTICOS
- 4 ALGUNOS PROGRAMAS ARITMÉTICOS

INTRODUCCIÓN

- PROLOG incluye herramientas extra-lógicas que permiten realizar de forma eficiente todas las operaciones aritméticas habituales en cualquier lenguaje de programación.
- Estas herramientas son de dos tipos:
 - **Constantes y operadores aritméticos**, que sirven para **construir** expresiones aritméticas.
 - **Predicados aritméticos**, que sirven para **evaluar** las expresiones aritméticas construidas mediante los operadores aritméticos anteriores.
- En lo que sigue se introducen tanto los operadores como los predicados aritméticos, y se ilustra su funcionamiento mediante algunos ejemplos de **programas aritméticos**.

CONSTANTES Y OPERADORES ARITMÉTICOS

Constantes numéricas

Las habituales: enteros y reales en notación decimal o científica.

Operadores aritméticos

$X+Y$	suma de X e Y
$X-Y$	X menos Y
$X*Y$	producto de X por Y
X/Y	cociente real de la división de X por Y
$X//Y$	cociente entero de la división de X por Y
$X \bmod Y$	resto de la división entera de X por Y
$\text{abs}(X)$	valor absoluto de X
$\text{sqrt}(X)$	raíz cuadrada de X
$\log(X)$	logaritmo neperiano de X
.....	

Observación

Los operadores aritméticos permiten simplemente construir expresiones aritméticas, pero estas no son más que *términos compuestos* (algunos, como los seis primeros de más arriba, en notación infija) que no representan ningún valor.

Ejemplo

La expresión $3+5$ no es más que el término compuesto $+(3, 5)$ escrito en notación infija.

- “`?- 3+5.`”: ERROR, puesto que “+” **no es un predicado**.
- La consulta “`?- 3+5 = 8.`” devuelve `false`, dado que el término compuesto $3+5$, equivalente a $+(3, 5)$, **no es unificable** con el término constante 8.

Para poder **evaluar expresiones aritméticas** en PROLOG hay que utilizar los predicados aritméticos que se describen a continuación.

PREDICADOS ARITMÉTICOS

X is Y Si Y es una expresión aritmética, esta se evalúa y el resultado se intenta *unificar* con X (término cualquiera).

A la hora de usar este predicado hay que tener en cuenta las siguientes consideraciones:

- ❶ Su uso puede dar lugar a un error en los dos siguientes casos:
 - ❶ cuando la parte derecha no es una expresión aritmética:


```
?- X is a+1. % a es una constante no numérica
{ ... ERROR: ...}
```
 - ❷ cuando la parte derecha es una expresión aritmética pero no se puede evaluar (error de instanciación):


```
?- X is 4*Z. % Z es una variable sin valor concreto
{... ERROR: ...}
```
- ❷ Salvo en los casos anteriores, el resultado del predicado dependerá de si la parte izquierda (**tal cual aparece**) unifica o no con el resultado obtenido al evaluar la parte derecha.

Ejemplos (Uso del predicado aritmético `is`)

?- `X is sqrt(4)` . resultado: $X = 2.0$

% `sqrt(4)` da 2, que se puede unificar (y se unifica) con X mediante u.m.g. $\{X = 2\}$

?- `5 is 2+3` . resultado: true

% `2+3` da 5, que se puede unificar con 5 con u.m.g. vacío

?- `3+5 is 3+5` . resultado: false

% el `3+5` de la dcha da 8, que NO se puede unificar con el `3+5=+(3,5)` de la izqda

?- `X is 5, Y is X+1` . resultado: $X = 5, Y = 6$

% PROLOG construye el Árbol de Resolución siguiente:

?- X is 5, Y is $X + 1$.

$\left| \sigma_1 = \{X/5\} \right.$

?- Y is 5 + 1.

$\left| \sigma_2 = \{Y/6\} \right.$

?-

$X_{\sigma_1\sigma_2} = \underline{5}_{\sigma_2} = 5, \quad Y_{\sigma_1\sigma_2} = \underline{Y}_{\sigma_2} = 6$
--

Otros predicados aritméticos, en los que, a diferencia de `is`, *ambos* argumentos deben ser expresiones aritméticas evaluables (se producirá un error si esto no es así). **Una vez evaluados X e Y:**

$X = Y$	cierto si los valores numéricos de X e Y son iguales
$X \neq Y$	cierto si los valores numéricos de X e Y son distintos
$X < Y$	cierto si el valor numérico de X es menor que el de Y
$X \leq Y$	cierto si el valor numérico de X es menor o igual que el de Y
$X > Y$	cierto si el valor numérico de X es mayor que el de Y
$X \geq Y$	cierto si el valor numérico de X es mayor o igual que el de Y

Ejemplos (Uso de predicados aritméticos)

```
?- X+3 < sqrt(4) .
.. ERROR: ..
```

```
?- 3+5 =< 8.
true (OJO, es =< y no <=)
```

```
?- 1+5 > abs(-8) .
false
```

```
?- 3 =\= 3*a.
.. ERROR: ..
```


Observación

Conviene resaltar las diferencias entre los siguientes predicados predefinidos:

$X = Y (X \backslash = Y)$	cierto si X e Y son términos que son (no son) unificables
$X ::= Y (X = \backslash = Y)$	cierto si X e Y son expresiones aritméticas, se pueden evaluar, y sus valores numéricos son (no son) iguales.
$X == Y (X \backslash == Y)$	cierto si X e Y son términos que son (no son) literalmente idénticos.

Ejercicios (Uso básico de operadores y predicados aritméticos)

Ejercicio nº 1 de la *Práctica de PROLOG nº 2*.

ALGUNOS PROGRAMAS ARITMÉTICOS

- En el apartado relativo a la **sintaxis de PROLOG** se ha discutido la implementación de algunos predicados aritméticos (suma, producto, par, impar) con lógica “pura”.
- Son programas interesantes por su *versatilidad* (por ejemplo el predicado `suma` también sirve para restar o para descomponer un natural en sumandos) pero *incómodos* -uso de `s(X)` para representar a los naturales- e *ineficientes* -cálculo recursivo-.
- En la práctica, para realizar programas aritméticos se deben utilizar los operadores y predicados aritméticos de PROLOG.
- Lo anterior supone ganar en comodidad y eficiencia pero perder la versatilidad: observe por ejemplo cómo el programa para sumar que se incluye a continuación, debido al uso de `is`, requiere que sus dos primeros argumentos sean de *entrada*, por lo que ya no sirve ni para restar ni para descomponer.

Ejemplo (Programas aritméticos “suma, producto” y “par”)

```
% suma(+X,+Y,?Z)
```

```
% cierto si Z es la suma de X e Y
```

```
    suma(X,Y,Z) :-
```

```
        Z is X+Y.
```

```
% producto(+X,+Y,?Z)
```

```
% cierto si Z es el producto de X e Y
```

```
    producto(X,Y,Z) :-
```

```
        Z is X*Y.
```

```
% par(+X)
```

```
% cierto si X es par
```

```
    par(X) :-
```

```
        X mod 2 == 0.
```

Ejemplo (Algunas consultas a “suma” y “par”)

- `?- suma(0, 1, 1) . True`
- `?- suma(1, 1, X) . X=2`
- `?- suma(1, 1, X), suma(X, 2, Z) . X=2, Z=4`
- `?- suma(0, 0, Z), suma(0, 2, Z) . False`
- `?- suma(1, Y, 3) . ERROR de instanciación`

Esta versión de “suma” NO sirve para restar.

- `?- suma(X, Y, 2) . ERROR de instanciación`
- `?- suma(X, Y, Z) . ERROR de instanciación`

Esta versión de “suma” NO sirve para descomponer.

- `?- par(4) . True`
- `?- par(X) . ERROR de instanciación`

Esta versión de “par” NO sirve para generar números pares.

Ejemplo (Programa aritmético para calcular factoriales)

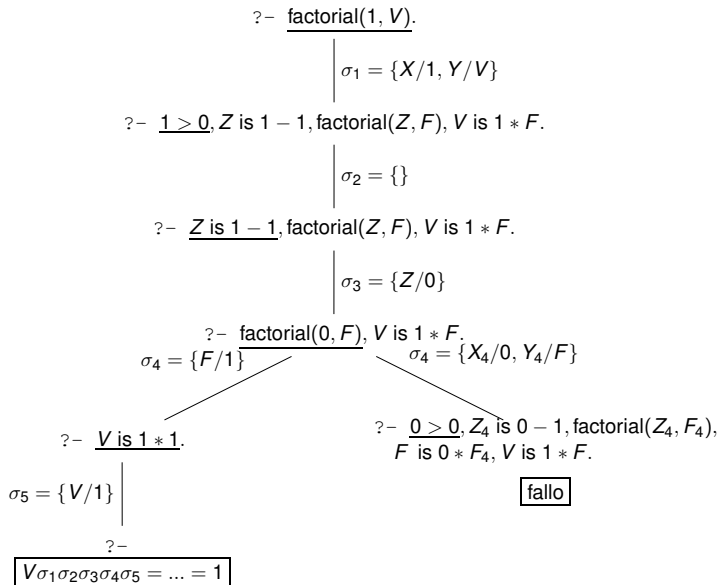
```
% factorial(+X, ?Y)
% cierto si Y es el factorial de X
```

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

```
factorial(X, Y) :-
    X > 0,
    Z is X-1,
    factorial(Z, F),
    Y is X*F.
```

Ejemplo (Algunas consultas a “factorial”)

- ?- factorial(1, 1) . *True*
- ?- factorial(4, X) . *X = 24*
- ?- factorial(X, 24) . *ERROR de instanciación*



Respuesta PROLOG: $V=1$ y false (no hay más soluciones)

Ejercicios (Cálculo de factoriales)

Ejercicio nº 2 de la *Práctica de PROLOG nº 2*.

Observación (Recursión final o de cola)

- La recursión de la implementación anterior de `factorial` es *recursión no final*: se realizan computaciones (en este caso una operación `is`) con posterioridad a la llamada recursiva.
- Por razones de eficiencia conviene utilizar siempre que se pueda **recursión final (o recursión de cola)**, aquella en la que la llamada recursiva es lo último que se computa.
- Una técnica habitual para transformar una recursión no final en final es añadir uno o más parámetros adicionales, denominados **parámetros de acumulación**, en los que se irán acumulando los valores parciales necesarios en cada llamada recursiva.

Ejemplo (Factorial con recursión de cola, 1/2)

```
% factorial_rc(+X, ?Y)
% cierto si Y es el factorial de X.
% Implementación con recursión de cola.

% sobrecarga del predicado añadiendo -en medio-
% un parámetro de acumulación cuyo valor inicial
% es el del caso base (el factorial de 0, 1).

factorial_rc(X, Y) :-
    factorial(X, 1, Y).

% caso base
% la salida es el valor acumulado

factorial(0, Ac, Ac).
```


Ejemplo (Factorial con recursión de cola, 2/2)

```
% caso recursivo
% actualización del parámetro de acumulación
% previa a la llamada recursiva
factorial(X, Ac, Y) :-
    X > 0,
    Z is X-1,
    NAc is X*Ac,    % actualización parámetro
    factorial(Z, NAc, Y).
```

Ejercicios (Programas aritméticos)

- ❶ *Construya el Árbol de Resolución para la consulta
?- factorial_rc(1, F) (con la implementación del predicado
factorial mediante recursión de cola).*
- ❷ *Ejercicios nº 3, 4 y 5 de la **Práctica de PROLOG nº2**.*

Soluciones propuestas:

?- factorial_rc(1, V).

$\sigma_1 = \{X/1, Y/V\}$

?- factorial(1, 1, V).

$\sigma_2 = \{X2/1, A2/1, Y2/V\}$

?- 1 > 0, Z2 is 1 - 1, NA2 is 1 * 1, factorial(Z2, NA2, V).

$\sigma_3 = \{\}$

?- Z2 is 1 - 1, NA2 is 1 * 1, factorial(Z2, NA2, V).

$\sigma_4 = \{Z2/0\}$

?- NA2 is 1 * 1, factorial(0, NA2, V).

$\sigma_5 = \{NA2/1\}$

?- factorial(0, 1, V).

$\sigma_6 = \{Y6/1, V/1\}$

$\sigma_6 = \{X6/0, A6/1, Y6/V\}$

?-

?- 0 > 0, Z6 is 0 - 1, NA6 is 0 * 1, factorial(Z6, NA6, V).

$V\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6 = \dots = 1$

fallo

BIBLIOGRAFÍA

- L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Mass., second edition, 1994.
- W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, fifth edition, 2003.
- I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition, 2001.
- J. Lloyd. *Foundations of Logic Programming*, (Second Edition). Springer-Verlag, 1987.
- R. O'Keefe. *The Craft of Prolog*. The MIT Press, Cambridge, MA, 1990.
- U. Nilsson and J. Maluszynski. **Logic, Programming and Prolog**. John Wiley & Sons Ltd, 1996.
- **SWI-Prolog**, entorno de programación en Prolog de dominio público.
- **comp.lang.prolog. Faq**

© 2022 Ana Pradera Gómez

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,
disponible en

<https://creativecommons.org/licenses/by-sa/4.0/deed.es>