Conferencia 5: Programación Declarativa

M.Sc Dafne García de Armas

Curso 2018-2019

Tema: Aritmética de Prolog

Introducción

En esta conferencia se introducen extensiones del lenguaje Prolog que tienen en general diferentes modos de realización específicos en diferentes sistemas en uso. En lo que sigue los desarrollos se ejemplificarán con las especificaciones del SWI Prolog. Luego para una completa información de las extensiones que se estudiarán, el estudiante debe consultar la ayuda del SWI Prolog y, por supuesto, ¡probar y probar ejemplos en el intérprete!

Aunque Prolog es un lenguaje cuyo objetivo es la programación de soluciones de problemas en los que los procesos lógicos son esenciales, el lenguaje necesita, como todo lenguaje de programación, una aritmética y hay una aritmética de base disponible en cualquier realización de un intérprete.

En algunos ejemplos de programas se han empleado números en Prolog, y ello quiere decir que el intérprete de Prolog aprovecha la representación de números y las operaciones aritméticas básicas del hardware sobre el cual se instala para desarrollar una aritmética y definir a partir de ella funciones matemáticas que puedan ser llamadas desde un programa Prolog.

Operadores aritméticos

Un operador n-ario se define como el nombre de un predicado n-ario o de un símbolo de función o constructor n-ario. Ejemplos de operadores son las operaciones aritméticas. En Prolog el repertorio de dichas operaciones es limitado dado que Prolog es un lenguaje orientado a la manipulación simbólica. La siguiente tabla muestra las operaciones básicas disponibles en la mayoría de las implementaciones de Prolog y los símbolos de función asociados.

Operación	Símbolo de función
Suma	+
Resta	-
Multiplicación	*
División entera	div
Módulo (resto de la división entera)	mod
División	/

Tabla 1: Operaciones aritméticas básicas.

Con estos operadores se pueden construir expresiones aritméticas que no son otra cosa que términos. Ejemplos:

```
3 + 2 * 5
3 / 2 + 2 * 5
6 div 2
3 mod 2
Y + 10 / 2
```

Operadores para comparar expresiones aritméticas

Las expresiones aritméticas pueden compararse utilizando operadores los de comparación que se muestran en la Tabla 2.

Operación	Símbolo	Significado
Mayor que	>	E1 > E2 E1 es mayor que E2
Menor que	<	E1 < E2 E1 es menor que E2
Mayor o igual que	>=	E1 >= E2 E1 es mayor o igual que E2
Menor o igual que	=<	E1 =< E2 E1 es menor o igual que E2
Igual que	=:=	E1 =:= E2 E1 es igual que E2
Distinto que	=\=	E1 =\= E2 E1 no es igual que E2

Tabla 2: Operaciones de comparación de operaciones aritméticas.

Los operadores de comparación aritmética fuerzan la evaluación de las expresiones aritméticas, luego realizan la comparación de los resultados obtenidos y no producen la instanciación de las variables. En el momento de la evaluación todas las variables deben estar instanciadas.

Evaluación de expresiones aritméticas con predicados de comparación:
?- 3 + 2 * - 5 >= 0.
false.
?- 4 =:= 3 + 2.
False
?- 4 =\= 3 + 2.
true
?- 3 * X > 6.
ERROR: >/2: Arguments are not sufficiently instantiated

Predicado is:

Estas expresiones no se evalúan automáticamente y es preciso forzar su evaluación mediante el operador is/2. Dado un objetivo ":-Exp1 is Exp2", donde Exp1 puede ser una variable o un número y Exp2 una expresión que no puede contener variables en el momento de su evaluación. Este predicado triunfa si Exp1 es unificable con el número que resulta de evaluar la expresión Exp2. Tanto la parte izquierda como la derecha son expresiones aritméticas, que de contener variables deberán ser instanciadas en el momento de la ejecución del objetivo, cuyo efecto es forzar la evaluación de dichas expresiones y después intentar unificarlas. Si Expresión1 es una variable, quedará instanciada al valor que tome Expresion2. La evaluación de cada operación aritmética se realiza mediante procedimientos especiales, predefinidos (built-in) en el intérprete, que acceden directamente a las facilidades de cómputo de la máquina.

```
(<variable>|<constante>) is <función aritmética>
```

Si en la parte izquierda aparece una variable, is realiza una asignación. Si aparece una constante, se realiza una comparación.

Ejemplos: evaluación de las expresiones aritméticas anteriores.

```
:- X is 3 + 2 * 5
X = 13

:- X is 3 / 2 + 2 * 5
X = 11.5

:- X is 6 div 2
X = 3

:- X is 3 mod 2
X = 1

:- 2 is 2
True

:- 2+3 is 2+3
False

:- X is Y + 10 / 2
ERROR: is/2: Arguments are not sufficiently instantiated

:- 2+3 is Y
ERROR: is/2: Arguments are not sufficiently instantiated
```

En el último caso el mensaje de error se produce porque la variable Y no está instanciada en el momento en que se evalúa la expresión. El operador is/2 no puede utilizarse para instanciar una variable dentro de una operación, por ejemplo: ":- 40 is X + 10/2.". Tampoco se debe utilizar para actualizar el valor de una variable, por ejemplo: ":- X is X + 1.", vea que si X está instanciada a un valor dado lo que ocurre es un fracaso en la unificación del resultado del miembro derecho al ser comparado con el valor de X en el miembro izquierdo.

El predicado is/2 pudiera utilizarse para verificar la igualdad entre una constante numérica y una expresión aritmética. Sin embargo, como se puede apreciar en el siguiente ejemplo, no siempre se obtiene la respuesta esperada:

```
?- 1.0 is sin(pi/2).
true
?- 1 is sin(pi/2).
False
```

El segundo objetivo falla debido a que la expresión sin(pi/2) evalúa al número real 1.0 el cual no unifica con el entero 1. Por lo general, se recomienda usar is/2 con Exp1 como variable. Si se desea verificar la igualdad de expresiones se debe utilizar el predicado =:=/2.

```
?- 1 =:= sin(pi/2).
true
```

Diferencias (is, =:=, =, ==)

Veamos con ejemplos las diferencias entre is, =:=, = y ==.

:- 2*3 =:= 3+3
True

:- 2*3 is 3+3
False

:- 2*3 = 3+3
False

:- 2*3 == 3+3
False
//Nótese en este ejemplo como el único operador que fuerza a la evaluación de ambas expresiones es el =:=

:- 3 =:= 3
True
:- 3 is 3
True
:- 3 = 3
True
:- 3 == 3
True

//Todos pueden usarse como función identidad.

:-X =:= 3*3
ERROR: =:=/2: Arguments are not sufficiently instantiated

:- X is 3*3
X=9

:- X = 3*3
X=3*3

:- X == 3*3
false

//Notese como el =:= no admite variables no unificadas en sus expresiones, el = solo unifica el valor, no fuerza la ejecución de las expresiones y el == como solo pregunta sobre si son exactamente iguales retorna falso. El comportamiento correcto en este caso se logra empleando is.

Predicado	Relación	Sustitución de variables	Evaluación aritmética
`==`	`identidad`	NO	NO
`=`	`unificable`	SÍ (izq+der)	NO
`=:=`	`mismo valor`	NO	SÍ (izq+der)
`is`	`asignación / comparación`	SÍ (izq)	SÍ (der)

Definición de funciones aritméticas

Además de las funciones básicas anteriores, SWI-Prolog provee de otras funciones aritméticas de gran utilidad, que no requieren de su definición por parte del programador (ej. abs/1, sign/1, max/2, min/2, random/1, eval/1, etc.).

También es posible a un predicado en Prolog darle el rol de función aritmética, asumiendo su último argumento como el retorno de la función y el resto de los argumentos como la entrada. Para declarar un predicado como una función aritmética se utiliza el predicado arithmetic_function/1 que toma como argumento la cabeza del predicado que se desea declarar como función aritmética en notación Nombre/Aridad.

Ejemplo: Declarando el predicado mean/3, que verifica si C es el promedio de A y B, como una función aritmética:

```
:- arithmetic_function(mean/2).
mean(A,B,C):- C is (A+B)/2.
```

Luego, se podrían hacer las siguientes consultas al programa:

```
?- A is mean(4, 5).
A = 4.5

?- mean(4, 5, A).
A = 4.5
```

Caracterización de los operadores

Un lenguaje declarativo como Prolog que tiene entre sus fines la representación y el procesamiento de información textual debe incluir la posibilidad de introducir y definir operadores con la notación requerida para representar texto con determinados fines de lectura y procesamiento.

Las expresiones sintácticas válidas en Prolog se construyen en general con una *notación prefija* de los operadores, como, por ejemplo:

```
padre(luis, X).
```

[2,3] que es “azúcar sintáctico” para la representación prefija interna al intérprete de Prolog de la lista .(2, .(3, nil))).

Es decir el operador precede a sus argumentos, los cuales se escriben a continuación de éste, encerrada entre paréntesis y separado por comas. Esta es la forma estandarizada en que el intérprete recibe términos y cláusulas para su interpretación. Sin embargo, como se vio en los apartados anteriores, Prolog cuenta con operadores con una sintaxis que difiere de la estándar: una notación infija de operadores, con la cual se incrementa la facilidad de lectura de un programa. Si esto no fuera así sería muy complicado escribir las operaciones aritméticas, por ejemplo: 2*3+4*5 tendría que escribirse como +(*(2,3),*(4,5)).

Debe aclararse que Prolog admite para sus operadores las notación prefija, por lo que en la expresión is(X, (+(*(2,3),*(4,5)))) no tiene errores de sintaxis y X unifica con el valor que resulta de evaluar la expresión aritmética del segundo argumento, por tanto X = 26.

Por otra parte véase que las reglas (cláusulas con cabeza y cuerpo) de un programa, para facilitar su lectura como condicionales, tienen una representación infija de su operador principal (:-).

Ejemplo:

member(X [_|Y]) :- member(X,Y).

Que para el intérprete tiene la representación prefija:

:- (member(X [_|Y]), member(X,Y)).

Además si se realiza la lectura declarativa de un programa existen hechos y reglas que tratadas en notación infija son más fáciles de comprender.

Ejemplo:

```
a es_padre_de b.
X es_abuelo_de Y :- X es_padre_de Z, (Z es_padre_de Y ; Z es_madre_de Y).
```

Por tanto puede resultar conveniente introducir operadores con una notación infija o incluso sufija.

El lenguaje Prolog proporciona facilidades para realizar esta tarea, es decir, definir o redefinir operadores de forma diferente a la estándar. Para ello analizaremos las características que rigen a un operador.

Cada operador en Prolog está caracterizado, además de por su *nombre/aridad* , por su *precedencia* (o prioridad), por su *posición* (prefija, infija, sufija), y por su *asociatividad* con respecto a otros operadores.

Nombre
El nombre es un átomo o una combinación de caracteres especiales (`+, -, *, /, <, >, =, :, &, _, ~`). Cuando se elija un nombre concreto para un operador habrá que tener en cuenta que ciertos nombres están reservados (e.g., "`is`", "`=`", "`+`", "`:-`", etc.).

Precedencia
En un término pueden ocurrir diferentes operadores que dan lugar a diferentes subtérminos del término. Ejemplo:

```
3 + 2 * 5
```

Para una correcta interpretación del término es necesario establecer en qué orden deberán ser tomados estos operadores. El orden en que se toma un operador con respecto al resto de los operadores se denomina *precedencia*, la precedencia se denota en Prolog con un número entre 1 y 1200, que indica cómo debe interpretarse una expresión en la que no hay paréntesis. En una expresión sin parénesis, el operador con mayor número de precedencia se convierte en el *operador principal* de la expresión, el cual es visible en la notación prefija del término al ser el más externo. El empleo de paréntesis anula las reglas de precedencia. Al establecer la precedencia entre operadores se elimina cualquier ambigüedad en la lectura y por lo tanto se realiza la correcta interpretación de un término. Si analiza

la Tabla 3 puede remarcar que no hay ambigüedad para interpretar 3 + 2 * 5 como 3 + (2 * 5) que es lo que normalmente se hace en matemática, siendo el operador principal la suma.

Precedencia	Operador
500	+, -
400	*, /

Tabla 3: Precedencia de los operadores predefinidos +, -, * y /

Posición

Como ya se ha indicado un operador puede tener una notación prefija que es la usual cuando definimos predicados en Prolog y la estándar para el intérprete. Pero existen operadores unarios y binarios primitivos en Prolog que han sido definidos con una notación infija, prefija o sufija y además esta facilidad es dada también al programador.

La posición de un operador es una característica que se define empleando especificaciones de modo. Los especificadores de modo son "_f_" para operadores infijos, "_f" para operadores sufijos (también llamados posfijos) y "f_" para operadores prefijos. El símbolo "_" puede sustituirse según sea el caso por x o y como se explica más adelante y f denota el operador al cual se le está definiendo la posición.

Por ejemplo, +, - que son operadores binarios infijos de suma y resta respectivamente, son también definidos como operadores unarios prefijos, para denotar números positivos y negativos respectivamente (ver Tabla 4): +2, -2.5.

Posición	Operador
yfx	+, -
fy	+, -

Tabla 4: Posiciones infijas y prefijas de los operadores + y -.

Asociatividad

Además de su colocación prefija, infija o sufija con respecto a sus argumentos, es necesario especificar de un operador cuál es su *asociatividad* con respecto a los operadores principales de los términos que recibe como argumentos. La asociatividad se emplea para eliminar la ambigüedad de las expresiones en las que aparecen operadores con la misma precedencia. Por ejemplo, determinar cómo interpretar una expresión de la forma: "`16 / 4 / 2`". Si se interpreta como "(`16 / 4) / 2`" el resultado es 2, pero si se interpreta como "`16 / (4 / 2)`" el resultado es 8.

La asociatividad de un operador se expresa en Prolog al mismo tiempo que la propiedad de posición, empleando los especificadores de modo, que son átomos (constantes no numéricas) especiales:

Prefijo		Infijo			Sufijo	
fx	fy	xfx	xfy	yfx	xf	Yf

Tabla 5: Significado de los átomos especiales especificadores de modo.

Como ya se dijo, f denota un operador y x, y denotan sus argumentos. Una "x" significa que es argumento debe tener un número de precedencia estrictamente menor que el operador que se está especificando. Por otra parte una "y" indica que el argumento correspondiente puede tener una precedencia igual o menor. Se define la precedencia de un argumento como sigue: si el argumneto es

un término su precedencia es cero, a menos que esté encabezado por un operador (al que se ha asignado una precedencia), en cuyo caso su precedencia es la de ese operador. Si el argumento está encerrado entre paréntesis su precedencia es cero.

Especificador de modo	Asociatividad
fy	A la derecha
yf	A la izquierda
xfy	A la derecha
yfx	A la izquierda

Tabla 6: Asociatividad de los operadores.

Note que el caso de fx, xfx y xf el operador no es asociativo, mientras que si lo es en los restantes casos. Note además que yfy no se define pues esto conlleva a ambigüedades respecto a cómo asociar.

Ejemplos:

1. Como "`/`" tiene posición y asociatividad "`yfx`", "`16/ 4/ 2`" se interpreta como "`(16 / 4) / 2`", esto es, el operador "`/`" es asociativo a la izquierda.
2. El operador que define las reglas "`:-`" tiene el especificador de modo es "`xfx`" pues es un operador infijo que no es asociativo.
3. El operador aritmético de cambio de signo "`-`" y la negación "`\+`" suelen especificarse con asociatividad "`fy`" con lo que se hace posible la doble negación (de un número o de un objetivo), por ejemplo: `X is - -5, X = 5`.
4. La "`,`" que denota la conjunción en el cuerpo de una cláusula, tiene asociatividad "`xfy`", es decir es un operador infijo asociativo a la derecha.
5. El "`;`" que denota la disyunción en el cuerpo de una cláusula, tiene asociativita "`xfy`", es decir es un operador infijo asociativo a la derecha.

Con respecto a estos dos últimos operadores considere la siguiente cláusula:

```
abuelo(X,Y) :- padre(X,Z), madre(Z, Y); padre(Z,Y).
```

Vea que ambos operados tienen el mismo tipo de asociatividad pero que el operador "`,`" tiene precedencia 1000 y el operador "`;`" tiene precedencia 1100. Al reformular la cláusula en notación prefija el intérprete hará una interpretación incorrecta de la definición del predicado abuelo/2:

```
:-(abuelo(X, Y), ;(,(padre(X, Z), madre(Z, Y)), padre(Z, Y))).
```

Luego resulta necesario agregar paréntesis a la disyunción:

```
abuelo(X,Y) :- padre(X,Z), (madre(Z, Y); padre(Z,Y)).
```

La siguiente tabla resume las características de algunos operadores predefinidos en SWI Prolog.

Precedencia	Modo	Nombre
1200	xfx	:-, is
1200	fx	:-
1100	xfy	;
1000	xfy	,

900	fy	\+
700	xfx	<, =, =:=, =<, >=, \=
500	yfx	+, -
400	yfx	*, /
200	fy	+, -

Tabla 7: Características de algunos operadores en SWI Prolog.

Declaración y definición de operadores

Un usuario en un programa puede agregar nuevos operadores insertando un tipo especial de cláusulas llamadas directivas y la directiva específica para esto es:

```
:-op( Precedencia, Modo, Nombre).
```

donde `Precedencia` es la precedencia, `Modo` es el especificador de modo y `Nombre` es el nombre del operador. Varios operadores pueden definirse utilizando una misma directiva en la que los nombres de los operadores se suministran formando una lista. Una declaración de un operador debe aparecer antes de cualquier expresión que vaya a utilizarla.

Ejemplos de definiciones de operadores predefinidos en el SWI Prolog:

```
:- op(1200, xfx, [:-, is]).
:- op(1100, xfy, ';').
:- op(1000, xfy, ',').
:- op(500, yfx, [+, -]).
```

Ya se ha dicho que un operador puede tener definiciones alternas. Por ejemplo, el operador "`:-`" utilizado en objetivos como los anteriores tiene la siguiente definición:

```
:- op(1200, fx, [:-]).
```

Es claro que para definir un nuevo operador, el usuario ha de tener en cuenta tanto los operadores predefinidos en el intérprete Prolog que son utilizados por su programa, así como las definiciones que ya haya introducido para otros operadores en el propio programa. La definición de operadores es un poderoso instrumento para extender la sintaxis primitiva de Prolog. Habitualmente los operadores se utilizan como functores, que combinan objetos para formar estructuras y no para invocar acciones. Para ello es necesario asociarle una definición: asociarle un conjunto de cláusulas que determinen su significado.

Ejemplo1:

Transformemos las definiciones de member/2 y concatena/3 para que se puedan escribir estas relaciones usando la siguiente sintaxis:

```
Elemeno en Lista.
Concatenar Lista1 y Lista2 da Lista3.
```

Declaremos "`en`", "`concatenar`", "`y`" y "`da`" como operadores y definamos los predicados correspondientes:

```
%Operadores sobre listas.
:-op(350, xfx, en).
:-op(300, fx, concatenar).
:-op(250, yfx, [y, da]).

%Elemento en Lista.
E en [E|_].
E en [_|Resto] :- E en Resto.

%concatenar Lista1 y Lista2 da Lista3.
concatenar [] y L da L.
concatenar [X|Resto] y L1 da [X|L] :- concatenar Resto y L1 da L.
```

La declaración de los operadores contribuya a que la expresión “concatenar L1 y L2 da L3” sea equivalente a “concatenar(da(y(L1, L2), L3))”. Observe que esta formalización de las relaciones de pertenencia y concatenación nos permite expresar cuestiones al sistema en un lenguaje muy cercano al lenguaje natural.

Ejemplo2:

Una sintaxis más cercana al lenguaje natural para el hecho

```
abuelo(luis, darío).
```

sería:

```
luis es_abuelo_de darío.
```

Luego definiendo las relaciones familiares del programa familia con notación infija.

```
%Familia3
:-module(familia3, [padre/2,madre/2,abuelo/2,abuela/2]).
:- (op(700, xfy, [es_padre_de, es_madre_de, es_abuelo_de, es_abuela_de])).
a es_padre_de b.
a es_padre_de c.
b es_padre_de d.
b es_padre_de e.
c es_madre_de f.
g es_madre_de b.
h es_madre_de e.
X es_abuelo_de Y :- X es_padre_de Z, (Z es_padre_de Y ; Z es_madre_de Y).
X es_abuela_de Y :- X es_madre_de Z, (Z es_padre_de Y ; Z es_madre_de Y).
```

Conclusión

Se introdujeron los operadores aritméticos básicos y diversos operadores de comparación de expresiones aritméticas.

En Prolog la aritmética no es declarativa ya que los operadores aritméticos no son inversibles: todos sus argumentos deben estar instanciados en la llamada.

Un operador se caracteriza por su nombre/aridad, su precedencia, su posición y su asociatividad.

Se estudió el modo en que Prolog permite declarar nuevos operadores: empleando la directiva:

```
:-op( Precedencia, Modo, Nombre).
```

Además de declarar un nuevo operador hay que definir la semántica operacional del mismo.