Programación Declarativa: Lógica y Restricciones

El lenguaje de programación ISO-Prolog (Parte I)

Mari Carmen Suárez de Figueroa Baonza
mcsuarez@fi.upm.es

Contenidos

- Predicados para Tipos
- Aritmética
- Acceso a Estructuras
- Predicados Meta-Lógicos
- Comparación de Términos
- Input/Output

Predicados para Tipos

- Términos Prolog:
  - constante
    - átomo
    - número
      - entero
      - real
  - variable
  - estructura
- Predicados para tipos: tienen éxito o fallan, pero no producen error
  - integer(X)
  - float(X)
  - number(X)
  - atom(X) → X es un término constante (aridad 0) no numérico
  - atomic(X) → X es una constante (átomo o número)
  - compound(X) → X es una estructura

Predicados para Tipos: Ejemplos

- ?- atom(vacio).
  - Yes
- ?- integer(-3).
  - Yes
- ?- compound([a,b|Xs]).
  - Yes
- ?- compound(-3.14).
  - No
- ?- integer([1]).
  - No
- ?- X = 2, integer(X).
  - Yes
- ?- compound([X]).
  - Yes

Aritmética (I)

- Términos aritméticos
  - Un número es un término aritmético
  - Si f es un functor aritmético y X1,…,Xn son términos aritméticos, entonces f(X1,…,Xn) es un término aritmético
- Functores aritméticos
  - +, -, *, / (cociente), // (cociente entero), mod (módulo), etc.
- Una expresión aritmética sólo puede ser evaluada si no contiene variables libres. En otro caso aparece un error de evaluación
  - (3*X+Y)/Z → correcta si cuando se evalúan X, Y, y Z son términos aritméticos, en otro caso se produce un error
  - a+3*X → se produce un error (‘a’ no es un término aritmético)

Aritmética (II)

- Predicados aritméticos
  - <, >, =<, >=, =:= (igualdad aritmética), =\= (desigualdad aritmética), etc.
    - Ambos argumentos se evalúan y se comparan los resultados
  - Z is X
    - X, que debe ser un término aritmético, se evalúa y el resultado se unifica con Z
- Ejemplos: supongamos que X vale 3 e Y 4, y que Z es una variable libre
  - Y < X+1, X is Y+1, X =:= Y. → fallo
  - Y < a+1, X is Z+1, X =:= f(a). → error

Aritmética (III)

- Ejemplos:
  - ?- X is 4/2 + 3/7.
    - X = 2.42857
  - ?- X is 2*4, 2 is X//3.
    - X = 8
  - ?- X is 3, Y is X+4.
    - X = 3, Y = 7
  - ?- Y is X+4, X is 3.
    - Error (porque X está libre)
  - ?- 3+4 is 3+4.
    - no
    - La parte izquierda no unifica con 7 (resultado de evaluar la expresión)
  - X is X+1
    - Fracaso si X está instanciada en la llamada
    - Error aritmético si X está libre

El orden de los literales es relevante en el uso de predicados evaluables

Aritmética: Ejemplo 1

- plus(X,Y,Z) :- Z is X + Y
  - Sólo funciona en modo (in, in, out). X e Y deben ser términos aritméticos

- Otra implementación de plus/3
  - plus(X,Y,Z):- number(X),number(Y), Z is X + Y. %in-in-out
  - plus(X,Y,Z):- number(X),number(Z), Y is Z - X. %in-out-in
  - plus(X,Y,Z):- number(Y),number(Z), X is Z - Y. %out-in-in

- Predicado 'suma' entre enteros para cubrir el caso en el que los sumandos puedan no estar instanciados pero el resultado sí

aritmetica_plus.pl

Aritmética: Ejemplo 2

- Factorial usando aritmética de Peano
  - factorial(0,s(0)).
  - factorial(s(N),F):- factorial(N,F1), times(s(N),F1,F).
- Factorial usando aritmética Prolog
  - factorial(0,1).
  - factorial(N,F):-

    N > 0,

    N1 is N-1,

    factorial(N1,F1),

    F is F1*N.

factorial.pl

Aritmética: Ejercicio 1

- Sucesión de Fibonacci: 0,1,1,2,3,5,8,13,21, …
  - cada término, salvo los dos primeros, es la suma de los dos anteriores
- Definir el predicado fibonacci/2 (fibonacci(N,X)) que se verifique si X es el N-ésimo término de la sucesión de Fibonacci.
  - ?- fibonacci(6,X).
    - X = 8

fibonacci.pl

Aritmética: Ejercicio 2

- Definir lista_numeros/3 (lista_numeros(N,M,L)) que se verifica si L es la lista de los números entre N y M, ambos inclusive
  - ?- lista_numeros(3,5,L).
    - L = [3,4,5]
  - ?- lista_numeros(3,2,L).
    - no

lista_numeros.pl

Aritmética: Ejercicio 3

- Definir el predicado mcd/3 (mcd(X,Y,Z)) que se verifique si Z es el máximo común divisor de X e Y
  - ?- mcd(10,15,X).
    - X=5

Acceso a Estructuras

- Los meta-predicados de inspección de estructuras permiten:
  - Descomponer una estructura en sus componentes
  - Componer una estructura a partir de sus componentes
- Prolog proporciona 3 meta-predicados de inspección:
  - functor/3
  - arg/3
  - =../2

Acceso a Estructuras: functor/3

- functor(E, F, A): la estructura E tiene functor F y aridad A
  - E es un término compuesto f(X1,…,Xn) → F=f , A=n
  - F es el átomo f y A es el entero n → X=f(X1,…,Xn)

- Ejemplos:
  - ?- functor(padre(juan,jose),padre,2).
    - yes
  - ?- functor(libro(autor, titulo), N, A).
    - N = libro, A = 2
  - ?- functor(X, libro, 2).
    - X = libro(_G358, _G359)
  - ?- functor(p, N, A).
    - N = p, A = 0
  - ?- functor(X, p, 0).
    - X = p

Acceso a Estructuras: functor/3

- En el uso (+,+,+) se comporta como un test
  - ?- functor(t(X,a),t,2).
    - Yes
  - ?- functor(2+3*5-1,'-',2).
    - Yes
  - ?- functor(a,a,0).
    - Yes
  - ?- functor([x,y],'.',2).
    - Yes

Acceso a Estructuras: functor/3

- En el uso (+,-,-) se utiliza para obtener el functor principal de un término
  - ?- functor(punto(a,X),F,N).
    - F = punto
    - N = 2
  - ?- functor([A,f(X),Y],F,N).
    - F = '.'
    - N = 2
  - ?- functor([],F,N)
    - F = []
    - N = 0

Acceso a Estructuras: functor/3

- En el uso (-,+,+) se comporta como un generador único: se utiliza para generar una plantilla de estructura
  - ?- functor(T,punto,2).
    - T = punto(_,_)
  - ?- functor(T,'+',2).
    - T = _ + _
  - ?- functor(T,'+',4).
    - T = '+'(_,_,_,_)
  - ?- functor(T,a,0).
    - T = a

Acceso a Estructuras: arg/3

- arg(P, E, C): la estructura E tiene el componente C en la posición P (contando desde 1)
  - P es un entero positivo, E es un término compuesto → C unifica con el p-ésimo argumento de E
  - Los argumentos de numeran a partir de 1, de izquierda a derecha
  - Permite acceder a los argumentos de una estructura de forma compacta y en tiempo constante

- Ejemplos:
  - ?- _T=date(9,February,1947), arg(3,_T,X).
    - X = 1947
  - ?- arg(2, libro(autor, titulo), X).
    - X = titulo
  - ?- arg(N, libro(autor, titulo), autor).
    - N = 1

Acceso a Estructuras: arg/3

- En el uso (+,+,-) se utiliza para obtener el argumento i-ésimo de una estructura
  - ?- arg(3,arco(i,q2,q0),A).
    - A = q0
  - ?- arg(1,[a,b,c,d],A).
    - A = a
  - ?- arg(2,[a,b,c,d],A).
  - ?- arg(3,[a,b,c,d],A).

Acceso a Estructuras: arg/3

- En el uso (+,-,+) se utiliza para instanciar el argumento i-ésimo de una estructura
  - ?- arg(2,arco(i,Q,q0),q5).
    - Q = q5
  - ?- arg(1,[X,b,c,d],a).
    - X = a

Acceso a Estructuras: functor y arg. Ejemplo 1

- subTerm(X,Y): X es un subtérmino del término Y
  - subTerm(X,X). Cualquier término es subtérmino de si mismo
  - subTerm(X,Y):-
    compound(Y),
    functor(Y,F,N),
    subTerm(N,X,Y).

    X es un subtérmino de un término compuesto Y si es subtérmino de uno de los argumentos

    subTerm/3 comprueba iterativamente todos los argumentos
  - subTerm(N,X,Y):- Decrementa el contador (número de argumentos) y llama recursivamente a subTerm
    N>1, N1 is N-1, subTerm(N1,X,Y).
  - subTerm(N,X,Y):- Caso en el que X es un subtérmino del n-ésimo argumento de Y
    arg(N,Y,A), subTerm(X,A).

Acceso a Estructuras: functor y arg. Ejemplo 2

- add_arrays(X,Y,Z): Z es el resultado de sumar las matrices X e Y
  - add_arrays(A1,A2,A3):-
    functor(A1,array,N),
    functor(A2,array,N),
    functor(A3,array,N),
    add_elements(N,A1,A2,A3).
  - add_elements(0,_A1,_A2,_A3).
  - add_elements(I,A1,A2,A3):-
    arg(I,A1,X1), %% I > 0
    arg(I,A2,X2),
    arg(I,A3,X3),
    X3 is X1 + X2,
    I1 is I - 1,
    add_elements(I1,A1,A2,A3).

Se comprueba que los tres argumentos tienen el mismo nombre de functor y la misma aridad

Las matrices se recorren desde el final hasta el principio (para usar un solo índice, deteniéndose a 0), y se suman los elementos correspondientes de las matrices

Acceso a Estructuras: =../2

- X =.. Y (se lee X univ Y)
  - X es cualquier término Prolog
  - Y es una lista cuya cabeza es el átomo del functor principal de X y cuyo resto está formado por los argumentos de X
  - Transforma un término estructurado en una lista:
    - padre(juan, jose) =.. [padre, juan, jose]
- Soporta los usos (+,+), (-,+) y (+,-)
- El uso (-,-) genera un error
  - ?- A =.. B.
    - instantiation error

Acceso a Estructuras: =../2

- En el uso (+,+) se comporta como un test
  - ?- f(a, X, g(b,Y)) =.. [f, a, X , g(b,Y)].
    - Yes
  - ?- [a,b,c] =.. [‘.’, a, [b,c]].
    - Yes
  - ? 2+3*5-1 =.. [‘-’,2+3*5,1].
    - Yes
  - ?- a =.. [a].
    - Yes

Acceso a Estructuras: =../2

- En el uso (+,-) se utiliza para descomponer un término en sus componentes
  - ?- punto(2,3) =.. Xs.
    - Xs = [punto, 2, 3]
  - ?- [A,f(X),Y] =.. Xs.
    - Xs = ['.', A, [f(X), Y]] ;
  - ?- sin(X)*cos(X) + 3.14 =.. Xs.
    - Xs = [+, sin(X)*cos(X), 3.14] ;
  - ?- 6 =.. Xs.
    - Xs = [6]
  - ?- [] =.. Xs.
    - Xs = [[]]

Acceso a Estructuras: =../2

- En el uso (-,+) se comporta como un generador único. Se utiliza para componer un término a partir de sus componentes
  - ?- T =.. ['+',a+b+c,d].
    - T = a+b+c+d
  - ?- T =.. [arco,ej1,i,q3,q5].
    - T = arco(ej1, i, q3, q5)
  - ?- T =.. ['.', p,[a,k]].
    - T = [p, a, k]
  - ?- T =.. [fin].
    - T = fin

Acceso a Estructuras: =../2. Ejercicio

- Suponemos las siguientes figuras geométricas, donde los argumentos de las distintas figuras son números que indican sus dimensiones
  - cuadrado(lado); rectangulo(anchura, altura); triangulo(lado1, lado2, lado3); circulo(radio)
- Definir un predicado escala/3 (escala(+F,+K,?KF)) que multiplique cada dimensión de la figura F por el factor K, obteniendo la figura escalada KF
- Ejemplo:
  - ?- escala(rectangulo(3,5), 2, R).
    - R = rectangulo(6,10)
  - ?- escala(circulo(6.5),0.5,C).
    - C = circulo(3.25)

Conversión entre Strings y Átomos

- name(A,S)
  - A es el átomo/número cuyo nombre es la lista de caracteres ASCII S
  - ?- name(hello,S).
    - S = [104,101,108,108,111]
  - ?- name(A,[104,101,108,108,111]).
    - A = hello
  - ?- name(A,"hello").
    - A = hello

Predicados Meta-Lógicos (I)

- Se utilizan para examinar el estado de instanciación actual de un término:
  - var(Term): es cierto si Term es una variable libre (no instanciada)
    - ?- var(X), X = f(a). % éxito
    - ?- X = f(a), var(X). % fallo
  - nonvar(Term): es cierto si Term no es una variable libre
    - ?- X = f(Y), nonvar(X). % éxito
  - ground(Term): es cierto si Term no contiene variables libres (es un término básico)
    - ?- X = f(Y), ground(X). % fallo

Predicados Meta-Lógicos (II)

- is_list(Term): es cierto si Term está instanciado a una lista
  - Es decir, a la lista vacía [] ó a un término con funtor '.' y aridad 2, donde el segundo argumento es una lista
  - ?- is_list(a). % fallo
  - ?- is_list(X). % fallo
  - ?- is_list([a,b,c]). % éxito

Predicados Meta-Lógicos: Ejemplo

- length(Xs,N):-
  var(Xs), integer(N), length_num(N,Xs). **% Xs es una variable libre % modo out-in**
- length(Xs,N):-
  nonvar(Xs), length_list(Xs,N). **Xs no es una variable libre % modo in-out**

- length_num(0,[]).
- length_num(N,[_|Xs]):-
  N > 0, N1 is N - 1, length_num(N1,Xs).

- length_list([],0).
- length_list([X|Xs],N):-
  length_list(Xs,N1), N is N1 + 1.

Note: esta definición no es necesaria; en realidad el predicado length ya es reversible (aunque menos eficiente que length_num(N,L), cuando L es una variable)

Comparación de Términos

- La igualdad de términos puede determinarse de diferentes formas
  - El operador "=" es la propia unificación. Esto es, se unifican las variables de los términos que se comparan
  - El operador "==" no unifica las variables de los términos que se comparan. Por tanto, una variable (no ligada) sólo será igual a sí misma
  - T1 = T2
    - Es cierto si T1 y T2 pueden unificarse
  - T1 \= T2
    - Es cierto si T1 y T2 no pueden unificarse
  - T1 == T2
    - Es cierto si T1 y T2 son idénticos
  - T1 \== T2
    - Es cierto si T1 y T2 no son idénticos

Comparación de Términos: Ejemplos

- ?- a == a.
  - si
- ?- a == X.
  - no
- ?- X == Y.
  - no
- ?- X == X.
  - X = _G2
- f(X) == f(X).
  - si
- ?- f(X) == f(Y).
  - no

Ordenación de Términos No Básicos

- Orden alfabético/lexicográfico:
  - X @> Y, X @>= Y, X @< Y, X @=< Y
  - P. ej: T1 @< T2 se verifica si el término T1 es anterior que T2 en el orden de términos de Prolog
  - Ejemplos:
    - ?- f(a) @> f(b). % fallo
    - ?- f(b) @> f(a). % éxito
    - ?- f(X) @> f(Y). % dependiente de la implementación
    - ?- X @< 3. => Yes
    - ?- ab @< ac. => Yes
    - ?- 21 @< 123. => Yes
    - ?- 12 @< a. => Yes
    - ?- g @< f(b). => Yes
    - ?- f(b) @< f(a,b). => Yes
    - ?- [a,1] @< [a,3]. => Yes
    - ?- [a] @< [a,3]. => Yes

Comparación de Términos No Básicos: Ejemplos

- subterm/2 con términos no básicos
  - subterm(Sub,Term):- Sub == Term. % Sub y Term son idénticos
  - subterm(Sub,Term):-

    nonvar(Term),

    functor(Term,F,N),

    subterm(N,Sub,Term). % subterm/3 no varía con respecto a la definición vista anteriormente
- insert/3 inserta un elemento en una lista ordenada
  - insert([], Item, [Item]).
  - insert([H|T], Item, [H|T]):- H == Item.
  - insert([H|T], Item, [Item, H|T]):- H @> Item.
  - insert([H|T], Item, [H|NewT]) :- H @< Item, insert(T, Item, NewT).

Entrada/Salida de Términos

- Predicado read(X):
  - Lee por teclado un término, que se instanciará en la variable X
  - El término debe ir seguido de "." y un carácter no imprimible (espacio o intro)
  - Se pueden introducir términos en minúsculas, o cadenas
- Predicado write(X):
  - Siempre se satisface; nunca se intenta resatisfacer
  - Si X está instanciada, se muestra en pantalla
  - Si no, se muestra la variable interna (e.g.,"_G244")
- Predicado nl:
  - Provoca un salto de línea
- Predicado tab(X):
  - Escribe X espacios en blanco
- Predicado display(X):
  - Muestra X sin interpretar los functores/operadores

Escritura de Términos: Ejemplo

- Escritura de una lista en una columna: escribir_columna/1
  - escribir_columna([]).
  - escribir_columna([Cabeza|Cola]):-
    write(Cabeza),
    nl,
    escribir_columna(Cola).

input-output.pl

Escritura/Lectura de Términos: Ejemplo

- Calcular y escribir el cubo de un número dado: cubo/0
  - cubo :-

    write('Siguiente item: '),
    read(X),
    procesa(X).

  - procesa(stop) :- !.

  - procesa(N) :-

    C is N*N*N,
    tab(3),
    write('El cubo de '),write(N),
    write(' is '), write(C), nl,
    cubo.

input-output.pl

Entrada/Salida de Caracteres

- Predicado get_code(X):
  - Si X no está instanciada, captura el primer carácter imprimible y lo instancia en X
  - Si X está instanciada, intenta hacer equiparación con la entrada
    - ?- get_code(X).

      |: d

      X= 100
    - ?- X=3, get_code(X).

      |: a

      no
- Predicado put_code(X):
  - Si X está instanciada a un código ASCII (entero positivo), entonces escribe el correspondiente caracter
    - ?- put_code (104).

      h

Entrada/Salida de Ficheros (I)

- Predicado see(X):
  - Establece como canal de entrada el fichero X
  - Si X no esta instanciada, se produce un error
- Predicado seeing(X):
  - Averigua el canal de entrada activo
- Predicado seen:
  - Cierra el fichero y restablece el teclado (*user*) como canal de entrada

- Nota: los ficheros se representan como átomos de Prolog, escribiéndolos entre comillas simples.
  - '/home/usuario/fichero.txt'
  - 'salida.txt'

Entrada/Salida de Ficheros (II)

- Predicado tell(X):
  - Establece como canal de salida el fichero X
  - Si X no esta instanciada, se produce un error
- Predicado telling(X):
  - Averigua el canal de salida activo
- Predicado told:
  - Cierra el fichero y restablece el teclado como canal de salida

input-output.pl (write_list_to_file)

Ejercicio

- Escribir un predicado (barras/1) que tenga el siguiente comportamiento:
  - ?- barras([1,2,5,3,4]).

    *

    **

    *****

    ***

    ****

    yes

Programación Declarativa: Lógica y Restricciones

El lenguaje de programación ISO-Prolog (Parte I)

Mari Carmen Suárez de Figueroa Baonza
mcsuarez@fi.upm.es