Programación Declarativa: Lógica y Restricciones

El lenguaje de programación ISO-Prolog (Parte I)

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

nidos

- dicados para Tipos
- mética
- eso a Estructuras
- dicados Meta-Lógicos
- nparación de Términos
- ut/Output

cados para Tipos

ninos Prolog:

nstante

átomo

número

- entero
- real

riable

tructura

licados para tipos: tienen éxito o fallan, pero no ducen error

eger(X)

at(X)

mber(X)

om(X) → X es un término constante (aridad 0) no numérico

omic(X) → X es una constante (átomo o número)

mpound(X) → X es una estructura

cados para Tipos: Ejemplos

:om(vacio).

s

teger(-3).

s

ompound([a,b|Xs]).

s

ompound(-3.14).

teger([1]).

= 2, integer(X).

s

ompound([X]).

s

ética (I)

inos aritméticos

número es un término aritmético

es un functor aritmético y X1,…,Xn son términos aritméticos, onces f(X1,…,Xn) es un término aritmético

tores aritméticos

*, / (cociente), // (cociente entero), mod (módulo), etc.

expresión aritmética sólo puede ser evaluada si no ene variables libres. En otro caso aparece un error valuación

X+Y)/Z → correcta si cuando se evalúan X, Y, y Z son términos méticos, en otro caso se produce un error

*X → se produce un error (‘a’ no es un término aritmético)

ética (II)

licados aritméticos

>, =<, >=, =:= (igualdad aritmética), =\= (desigualdad aritmética),
.

Ambos argumentos se evalúan y se comparan los resultados

s X

X, que debe ser un término aritmético, se evalúa y el resultado se unifica con Z

nplos: supongamos que X vale 3 e Y 4, y que Z es variable libre

X+1, X is Y+1, X =:= Y. → fallo

a+1, X is Z+1, X =:= f(a). → error

ética (III)

nplos:

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)

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

3+4 is 3+4.

no

La parte izquierda no unifica con 7 (resultado de evaluar la expresión)

s X+1

Fracaso si X está instanciada en la llamada

Error aritmético si X está libre

ética: Ejemplo 1

(X,Y,Z) :- Z is X + Y

lo funciona en modo (in, in, out). X e Y deben ser términos tméticos

implementación de plus/3

```
s(X,Y,Z):- number(X),number(Y), Z is X + Y.      %in-in-out
s(X,Y,Z):- number(X),number(Z), Y is Z - X.      %in-out-in
s(X,Y,Z):- number(Y),number(Z), X is Z - Y.      %out-in-in
```

licado 'suma' entre enteros para cubrir el caso en el los sumandos puedan no estar instanciados pero el ltado sí

plus.pl

ética: Ejemplo 2

orial usando aritmética de Peano

```
torial(0,s(0)).
torial(s(N),F):- factorial(N,F1), times(s(N),F1,F).
```

orial usando aritmética Prolog

```
torial(0,1).
torial(N,F):-
 > 0,
1 is N-1,
actorial(N1,F1),
is F1*N.
```

/

ética: Ejercicio 1

esión de Fibonacci: 0,1,1,2,3,5,8,13,21, …

da término, salvo los dos primeros, es la suma de los dos teriores

nir el predicado fibonacci/2 (fibonacci(N,X)) que se fique si X es el N-ésimo término de la sucesión de nacci.

fibonacci(6,X).

X = 8

o/

ética: Ejercicio 2

nir lista_numeros/3 (lista_numeros(N,M,L)) que se fica si L es la lista de los números entre N y M, os inclusive

ista_numeros(3,5,L).

L = [3,4,5]

ista_numeros(3,2,L).

no

eros.pl

ética: Ejercicio 3

nir el predicado mcd/3 (mcd(X,Y,Z)) que se verifique
es el máximo común divisor de X e Y

mcd(10,15,X).

X=5

o a Estructuras

meta-predicados de inspección de estructuras
niten:

scomponer una estructura en sus componentes

mponer una estructura a partir de sus componentes

og proporciona 3 meta-predicados de inspección:

nctor/3

g/3

/2

o a Estructuras: functor/3

tor(E, F, A): la estructura E tiene functor F y aridad A

s un término compuesto f(X1,…,Xn) → F=f , A=n

s el átomo f y A es el entero n → X=f(X1,…,Xn)

nplos:

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

o a Estructuras: functor/3

l 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
```

o a Estructuras: functor/3

el uso (+,-,-) se utiliza para obtener el functor cipal 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

o a Estructuras: functor/3

l uso (-,+,+) se comporta como un generador único:
tiliza 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

o a Estructuras: arg/3

P, E, C): la estructura E tiene el componente C en la
ción P (contando desde 1)

es un entero positivo, E es un término compuesto → C unifica
n el p-ésimo argumento de E

s argumentos de numeran a partir de 1, de izquierda a derecha

rmite acceder a los argumentos de una estructura de forma
mpacta y en tiempo constante

nplos:

_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

o a Estructuras: arg/3

l uso (+,+,-) se utiliza para obtener el argumento i-
no 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).

o a Estructuras: arg/3

l uso (+,-,+) se utiliza para instanciar el argumento i-
no de una estructura

arg(2,arco(i,Q,q0),q5).

Q = q5

arg(1,[X,b,c,d],a).

X = a

a Estructuras: functor y arg. Ejemplo 1

erm(X,Y): X es un subtérmino del término Y

```
bTerm(X,X).
```

Cualquier término es subtérmino de si mismo

```
bTerm(X,Y):-
ompound(Y),
unctor(Y,F,N),
ubTerm(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

```
bTerm(N,X,Y):-
>1, N1 is N-1, subTerm(N1,X,Y).
```

Decrementa el contador (número de argumentos) y llama recursivamente a subTerm

```
bTerm(N,X,Y):-
rg(N,Y,A), subTerm(X,A).
```

Caso en el que X es un subtérmino del n-ésimo argumento de Y

a Estructuras: functor y arg. Ejemplo 2

_arrays(X,Y,Z): Z es el resultado de sumar las rices X e Y

```
d_arrays(A1,A2,A3):-
unctor(A1,array,N),
unctor(A2,array,N),
unctor(A3,array,N),
dd_elements(N,A1,A2,A3).
d_elements(0,_A1,_A2,_A3).
d_elements(I,A1,A2,A3):-
rg(I,A1,X1), %% I > 0
rg(I,A2,X2),
rg(I,A3,X3),
3 is X1 + X2,
1 is I - 1,
dd_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

o a Estructuras: =../2

Y (se lee X univ Y)

s cualquier término Prolog

s una lista cuya cabeza es el átomo del functor principal de X y
yo resto está formado por los argumentos de X

nsforma un término estructurado en una lista:

padre(juan, jose) =.. [padre, juan, jose]

orta los usos (+,+), (-,+) y (+,-)

so (-,-) genera un error

A =.. B.

instantiation error

o a Estructuras: =../2

l 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

+3*5-1 =.. [‘-’,2+3*5,1].

Yes

a =.. [a].

Yes

o a Estructuras: =../2

:l uso (+,-) se utiliza para descomponer un término
us 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 = [[]]

o a Estructuras: =../2

el uso (-,+) se comporta como un generador único. utiliza para componer un término a partir de sus ponentes

```
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
```

o a Estructuras: =../2. Ejercicio

onemos las siguientes figuras geométricas, donde rgumentos de las distintas figuras son números que can sus dimensiones

adrado(lado); rectangulo(anchura, altura); triangulo(lado1, o2, lado3); circulo(radio)

nir un predicado escala/3 (escala(+F,+K,?KF)) que tiplique cada dimensión de la figura F por el factor bteniendo la figura escalada KF

nplo:

escala(rectangulo(3,5), 2, R).

R = rectangulo(6,10)

escala(circulo(6.5),0.5,C).

C = circulo(3.25)

ersión entre Strings y Átomos

e(A,S)

es el átomo/número cuyo nombre es la lista de caracteres ASCII

name(hello,S).

S = [104,101,108,108,111]

name(A,[104,101,108,108,111]).

A = hello

name(A,"hello").

A = hello

cados Meta-Lógicos (I)

tilizan para examinar el estado de instanciación
l de un término:

Term): es cierto si Term es una variable libre (no instanciada)

- var(X), X = f(a). % éxito
- X = f(a), var(X). % fallo

var(Term): es cierto si Term no es una variable libre

- X = f(Y), nonvar(X). % éxito

und(Term): es cierto si Term no contiene variables libres (es un
mino básico)

- X = f(Y), ground(X). % fallo

cados Meta-Lógicos (II)

ist(Term): es cierto si Term está instanciado a una lista

s decir, a la lista vacía [] ó a un término con funtor '.' y aridad 2, donde el egundo argumento es una lista

- is_list(a). % fallo
- is_list(X). % fallo
- is_list([a,b,c]). % éxito

cados Meta-Lógicos: Ejemplo

```
h(Xs,N):-
ar(Xs), integer(N), length_num(N,Xs).   % Xs es una variable libre
                                        % modo out-in
h(Xs,N):-
onvar(Xs), length_list(Xs,N).          % Xs no es una variable libre
                                        % modo in-out

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

h_list([],0).
h_list([X|Xs],N):-
ength_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)

aración de Términos

gualdad de términos puede determinarse de entes formas

perador "=" es la propia unificación. Esto es, se unifican las ables de los términos que se comparan

perador "==" no unifica las variables de los términos que se nparan. Por tanto, una variable (no ligada) sólo será igual a sí ma

= T2

s cierto si T1 y T2 pueden unificarse

= T2

s cierto si T1 y T2 no pueden unificarse

== T2

s cierto si T1 y T2 son idénticos

== T2

s cierto si T1 y T2 no son idénticos

aración de Términos: Ejemplos

:= a.

:= X.

:= Y.

:= X.

_G2

= f(X).

) == f(Y).

ación de Términos No Básicos

n alfabético/lexicográfico:

> Y, X @>= Y, X @< Y, X @=< Y

j: T1 @< T2 se verifica si el término T1 es anterior que T2 en el
en de términos de Prolog

mplos:

- 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

ıración de Términos No Básicos: Ejemplos

erm/2 con términos no básicos

```
term(Sub,Term):- Sub == Term.    % Sub y Term son idénticos
term(Sub,Term):-
onvar(Term),
unctor(Term,F,N),
ubterm(N,Sub,Term).    % subterm/3 no varía con respecto a la definición vista anteriormente
```

t/3 inserta un elemento en una lista ordenada

```
rt([], Item, [Item]).
rt([H|T], Item, [H|T]):- H == Item.
rt([H|T], Item, [Item, H|T]):- H @> Item.
rt([H|T], Item, [H|NewT]) :- H @< Item, insert(T, Item, NewT).
```

da/Salida de Términos

cado read(X):

por teclado un término, que se instanciará en la variable X

término debe ir seguido de "." y un carácter no imprimible
pacio o intro)

ueden introducir términos en minúsculas, o cadenas

cado write(X):

mpre se satisface; nunca se intenta resatisfacer

está instanciada, se muestra en pantalla

o, se muestra la variable interna (e.g.,"_G244")

cado nl:

voca un salto de línea

cado tab(X):

ribe X espacios en blanco

cado display(X):

estra X sin interpretar los functores/operadores

ura de Términos: Ejemplo

:ura de una lista en una columna: escribir_columna/1

ribir_columna([]).

ribir_columna([Cabeza|Cola]):-

vrite(Cabeza),

l,

scribir_columna(Cola).

ut.pl

ura/Lectura de Términos: Ejemplo

lar y escribir el cubo de un número dado: cubo/0

```
o :-
rite('Siguiente item: '),
ead(X),
rocesa(X).

cesa(stop) :- !.

cesa(N) :-
 is N*N*N,
ab(3),
rite('El cubo de '),write(N),
rite(' is '), write(C), nl,
ubo.
```

ut.pl

da/Salida de Caracteres

cado get_code(X):

no está instanciada, captura el primer carácter imprimible y lo ancia en X

está instanciada, intenta hacer equiparación con la entrada

```
- get_code(X).
: d
= 100
- X=3, get_code(X).
: a
o
```

cado put_code(X):

está instanciada a un código ASCII (entero positivo), entonces ribe el correspondiente caracter

```
- put_code (104).
```

da/Salida de Ficheros (I)

cado see(X):

ablece como canal de entrada el fichero X

no esta instanciada, se produce un error

cado seeing(X):

rigua el canal de entrada activo

cado seen:

ra el fichero y restablece el teclado (*user*) como canal de entrada

: los ficheros se representan como átomos de Prolog, biéndolos entre comillas simples.

me/usuario/fichero.txt'

da.txt'

da/Salida de Ficheros (II)

cado tell(X):

ablece como canal de salida el fichero X

no esta instanciada, se produce un error

cado telling(X):

rigua el canal de salida activo

cado told:

ra el fichero y restablece el teclado como canal de salida

ut.pl (write_list_to_file)

cio

bir un predicado (barras/1) que tenga el siguiente
portamiento:

arras([1,2,5,3,4]).

*
****
**
***
es

Programación Declarativa: Lógica y Restricciones

El lenguaje de programación ISO-Prolog (Parte I)

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