Programación Declarativa: Lógica y Restricciones

l lenguaje de programación ISO-Prolog

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

nidos

dicados para Tipos

mética

eso a Estructuras

dicados Meta-Lógicos

nparación de Términos

rada/Salida

ta-Programación

dificación Dinámica

sing

ados para Tipos

ninos Prolog:

- nstante
  - átomo
  - número
    - entero
    - real
- riable
- tructura

dicados 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

ados 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

ados 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

ados 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
```

ados 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

ración 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).
```

la/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

la/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).
```

la/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'

la/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)

la/Salida: Ejercicio

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

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

*
****
**
***
es

ados de Orden Superior

emántica viene dada en términos de un lenguaje de
n inferior

predicados de 2º orden (meta-lenguaje) "hablan sobre" símbolos
un lenguaje de 1er orden (lenguaje-objeto)

de predicados de orden superior:

nprobación de tipos: integer/1, atom/1, var/1, ground/1

strucción de fórmulas: =../2

ución y control de la ejecución de objetivos: call/1

opilación de múltiples soluciones: findall/3, setof/3

n predicados de orden superior:

ellos que permiten añadir o eliminar cláusulas del conjunto soporte
grama) en tiempo de ejecución (programación dinámica): assert/1,
act/1, instance/2, etc.

orte !/0 que es un mecanismo de control del *backtraking* cuya
axis y ejecución es la de un predicado sin serlo realmente

Programación

rgumentos de los predicados “ordinarios” cumplen amente dos funciones:

tienen datos sobre los que razonar

g. member(2,[1,2,3])

tienen variables a instanciar

g. plus(2,3,X)

neta-predicado tiene entre sus argumentos otros a) predicados cuya prueba es parte de la prueba del -predicado

```
ional(Goal):- call(Goal).
ional(_Goal).
```

all/1 (predefinido) tiene éxito cuando su argumento lo tiene

j. `?- member(c,[a,b]).` *versus* `?- opcional(member(c,[a,b])).`

Programación: call/1

eta-predicado call(X) convierte el término X en un
ivo y llama a dicho objetivo

ebe estar instanciado a un término, sino se produce un error

) se cumple si se satisface X como objetivo

a habitualmente para

ta-programación (intérpretes, *shells*)

inir negación

lementar orden superior

plo:

).

) :- call(X).

- p(q(Y)).

= a

Programación: call/1. Ejemplos

Nota: este ejemplo carece de utilidad puesto que se puede ejecutar la llamada a mipred/1 directamente

plo:

```
red(X) :- display(X), nl.          % Un predicado
mplo:- X = mipred(5), call(X).     % Llamada de orden superior
```

plo: call/1 resulta muy útil en combinación con otros cados como univ/2

```
eto(12).
eto(13).
eto(78).
car(Predicado):-
ujeto(X), LLamada =.. [Predicado,X], call(LLamada), nl, fail.
car(_).
```

ramacion.pl

ado fail/0

redicado fail/0 es un predicado predefinido que
ore falla

mpre produce fallo

a cuando se ejecuta (similar al objetivo a=b)

n objetivo que nunca se satisface

iliza para detectar prematuramente combinaciones
s argumentos que no llevan a solución

ando la ejecución de código que va a fallar

il cuando queremos detectar casos explícitos que
dan un predicado

y fallo

evitar la aplicación de una regla, se puede forzar el con una combinación de *cut* y *fail*

plo: Comprobación de diferencia

erent(X,X) :- !, fail.

erent(X,Y).

mbinación de corte y fallo (*cut-fail*) permite forzar el so de un predicado

ecificando una respuesta negativa

pero hay que usarlo con cuidado

ramacion.pl

y fallo: Ejemplo

cado ground/1 para verificar que un término no ene variables libres (es un término básico)

a tan pronto se encuentra una variable libre

```
und(Term):- var(Term), !, fail.
und(Term):-
onvar(Term), functor(Term,F,N), ground(N,Term).

und(0,T).          % se han recorrido todos los subtérminos
und(N,T):-
>0, arg(N,T,Arg), ground(Arg), N1 is N-1, ground(N1,T).
```

rogramación: negación como fallo (I)

egación en Prolog consiste el predicado predefinido
gundo orden '\+'/1

ibe como argumento un objetivo

icho objetivo tiene éxito la negación falla y viceversa

mplo: \+ (X > 5) es equivalente a X =< 5

usa el corte y el predicado fail

(Goal) :- call(Goal), !, fail.

(Goal).

rminación de not(Goal) depende de la terminación
bal

(Goal) termina si se encuentra éxito para Goal antes de una rama
nita

(Goal) tiene éxito cuando Goal no puede ser probado

rogramación: negación como fallo (II)

ona de manera adecuada para objetivos básicos (no enen variables libres)

esponsabilidad del programador asegurar esta condición

nca instancia variables

l pero hay que saber utilizarlo:

narried_student(X):- not(married(X)), student(X).

dent(joe). ?- unmarried_student(joe).

ried(john). ?- unmarried_student(X).

g asume que aquellos objetivos que no tienen ión (fallan) son falsos

lquier cosa que no puede probarse con las reglas y los hechos de ase de conocimiento se considera falsa

ogramación: negación como fallo. Ejemplo

ición de conjuntos disjuntos:

```
rlap(S1,S2):-      % S1 y S2 se solapan si comparten algún elemento
nember(X,S1),member(X,S2).
oint(S1,S2):-
+overlap(S1,S2).
```

isjoint([a,b,c],[2,c,4]).

o

isjoint([a,b],[1,2,3,4]).

es

isjoint([a,c],X).

o

rogramación: negación como fallo. Ejemplo

```
bado(X) :- not(suspenso(X)), matriculado(X).
iculado(juan).
iculado(luis).
enso(juan).
```

ultas

probado(luis).

es

probado(X).

o

rogramación: negación como fallo. Ejemplo

```
bado2(X) :- matriculado2(X), not(suspenso2(X)).
iculado2(juan).
iculado2(luis).
enso2(juan).
```

¿Qué pasa si cambiamos el orden de los predicados?

?- aprobado2 (X).

pl

cio

ir el predicado borra/3 (borra(L1,X,L2)) que se ca si L2 es la lista obtenida eliminando los elementos unificables simultáneamente con X

ucion 1: definición con not

ucion 2: definición con corte

orra([a,b,a,c],a,L).
= [b, c] ;
o
orra([a,Y,a,c],a,L).
= a
= [c] ;
o
orra([a,Y,a,c],X,L).
= a
= a
= [c] ;
o

cio

ición con not

```
ra_1([],_,[]).
ra_1([X|L1],Y,L2) :-
=Y,
orra_1(L1,Y,L2).
ra_1([X|L1],Y,[X|L2]) :-
ot(X=Y),
orra_1(L1,Y,L2).
```

cio

ición con corte

```
ra_2([],_,[]).
ra_2([X|L1],Y,L2) :-
=Y, !,
orra_2(L1,Y,L2).
ra_2([X|L1],Y,[X|L2]) :-
 not(X=Y),
orra_2(L1,Y,L2).
```

predicados de control

neta-predicados de control se encargan de imponer
ol sobre sus argumentos:

```
_once(Goal):- call(Goal), !.
```

ugar de que ese control deba imponerse en la
ición de cada predicado a controlar:

```
_check(X,[X|_]):- !.
_check(X,[_|Y]):-
mber_check(X,Y).
```

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

```
check(2,[1,2,3,2,4]).   ?- just_once(member(2,[1,2,3,2,4])).
```

ajas de los meta-predicados de control:

- ntrol explícito y su definición centralizada en el meta-predicado
- s predicados a controlar mantienen múltiples usos y mayor
  aridad

predicados de control más frecuentes

ción como fallo:

```
not(Goal):- call(Goal), !, fail.
```

ba determinista:

```
just_once(Goal):- call(Goal), !.
```

ba condicional:

```
nelse(If,Then,_):-
        call(If),
        call(Then).

Then ; _Else:-
        call(If),
        !,
        call(Then).
```

```
ifthenelse(If,_Then,Else):-
        \+ If,
        call(Else).

If -> _Then ; Else:-
         call(Else).
```

:

```
while(Cond,Do):-
        call(Cond),
        call(Do),
        fail.
while(_,_).
```

```
while(Cond,Do):-
        call(Cond),
        ( Do -> fail ; (!, fail) ).

while(_,_).
```

Programación: findall/3

meta-predicado findall/3 (findall(Term, Goal, esults)) se verifica si ListResults es el conjunto de s las instancias del término Term que verifican el ivo Goal

Results es [] si no hay instancias de Term

úmero de soluciones debería ser finita (y enumerable en un npo finito)

plos:

ndall(X,(member(X,[d,4,a,3,d,4,2,3]),number(X)),L).

= [4, 3, 4, 2, 3]

ndall(X,(member(X,[d,4,a,3,d,4,2,3]),compound(X)),L).

= []

Programación: setof/3

eta-predicado setof/3 (setof(Term, Goal, ListResults))
erifica si ListResults es la lista ordenada sin
iciones de las instancias del término Term que
can el objetivo Goal

redicado falla si no hay instancias de Term

onjunto debe ser finito (y enumerable en tiempo finito)

plos:

etof(X,(member(X,[d,4,a,3,d,4,2,3]),number(X)),L).

= [2, 3, 4]

etof(X,member(X,[d,4,a,3,d,4,2,3]),L).

= [2, 3, 4, a, d]

etof(X,(member(X,[d,4,a,3,d,4,2,3]),compound(X)),L).

o

Programación: bagof/3

f/3 es igual que setof/3, pero devuelve una lista no
nada y con duplicados (según el orden del
tracking)

redicado falla si no hay instancias de Term

onjunto debe ser finito (y enumerable en tiempo finito)

plos:

agof(X,(member(X,[d,4,a,3,d,4,2,3]),number(X)),L).

= [4,3,4,2,3]

agof(X,member(X,[d,4,a,3,d,4,2,3]),L).

= [d,4,a,3,d,4,2,3]

agof(X,(member(X,[d,4,a,3,d,4,2,3]),compound(X)),L).

o

cio 1

enomina factor de un número natural N, a otro ero también natural que es divisor de N, pero ente de N

factores de 28 son 1, 2, 4, 7 y 14

ir el predicado factores/2 (factores(+N,-L)) que se que si L es la lista ordenada de los factores del ero N

actores(42,L).

= [1,2,3,6,7,14,21]

aturales.pl

cio 2

úmeros naturales se pueden clasificar en tres tipos:

s de tipo a si N es mayor que la suma de sus factores

s de tipo b si N es igual que la suma de sus factores

s de tipo c si N es menor que la suma de sus factores

ir el predicado tipoNatural/2 (tipoNatural(+N,-T))
e verifique si T es el número N

Natural(10,T).
=a

Natural(28,T).
=b

Natural(12,T).
=c

aturales.pl

cio 3

ir el predicado soloConsonantes/2
Consonantes(+P,-Q)) que se verifica si Q es la palabra
se obtiene al eliminar todas las vocales de la palabra

oloConsonantes(segoviano,P).

=sgvn

es.pl

cio 4

ir el predicado traduceDigitos/2
uceDigitos(+L1,-L2)) que se verifica si L2 es la lista de
ras correspondientes a los dígitos de la lista L1

raduceDigitos([1,2],L).
= [uno,dos]

ución 1: usando recursividad

ución 2: usando metapredicados

: usar el predicado auxiliar nombreDigito/2
breDigito(D,N)) que se verifica si N es el nombre del
D

gitos.pl

icación Dinámica (I)

ase de conocimientos en Prolog se puede modificar empo de ejecución (mientras se ejecuta el programa)

y potente

permite

dir conocimiento adquirido durante la ejecución

rimir reglas que se hacen innecesarias durante la ejecución

ular algunas técnicas de programación imperativa no disponibles ctamente en Prolog

unos trucos de programación

es, esto es muy útil, pero a menudo un error

igo difícil de leer, difícil de entender, difícil de depurar

camente, lento

icación Dinámica (II)

modificación dinámica debe utilizarse
rádicamente, con cuidado y a nivel local

a ello existen predicados que añaden o eliminan cláusulas de la
e de conocimientos

afirmación y la retracción pueden justificarse
amente en algunos casos:

rción de cláusulas que lógicamente se derivan del programa
nas)

racción de las cláusulas que son lógicamente redundante

mportamiento y/o los requisitos pueden diferir entre
nplementaciones de Prolog

lo general, el predicado debe declararse dinámico

dynamic predicado/n.

cación Dinámica: Añadir Conocimiento (I)

t/1 (assert(Clausula)): añade la cláusula a la base de cimientos al final de todas las cláusulas del predicado

usula debe estar instanciada a una cláusula Prolog

plo:

re(pepe, juan). Hecho en la base de conocimientos.

adre(pepe, X).

juan

ssert(padre(pepe, javi)), padre(pepe, X).

juan (;)

javi

cación Dinámica: Añadir Conocimiento (II)

t/1 (assert(Clausula)): (continuación)

e introduce una regla, hay que encerrarla entre paréntesis para
los distintos operadores que posea no se confundan

mplo:

```
ssert( (hijo(X, Y):- padre(Y, X) ) ).
```

ta(Clausula): como assert, pero coloca la cláusula en
er lugar

dmiten *backtracking*

cación Dinámica: Eliminar Conocimiento (I)

ct(Clausula): elimina de la base de conocimientos la era cláusula unificable con Clausula, que no debe ser variable

backtracking puede eliminar otras cláusulas

o hay cláusulas, falla

plo:

re(pepe, juan).

re(pepe, javi).

etract(padre(pepe, X)), padre(pepe, X).

javi

cación Dinámica: Eliminar Conocimiento (II)

sh(Predicado/Aridad): elimina todas las cláusulas del
cado Predicado con aridad Aridad

dicado debe estar instanciado

plo:

re(pepe, juan).

re(pepe, javi).

bolish(padre/2).

adre(X,Y).

tence error

icación Dinámica: Ejemplo 1

```
e_numbers(X, Y):- assert(related(X, Y)).
ate_numbers(X, Y):- retract(related(X, Y)).

ated(1, 2).

ate_numbers(1, 2).

ated(1, 2).

relate_numbers(1, 2).

ated(1, 2).
```

odificacionDinamica.pl

icación Dinámica: Ejemplo 2

eros de Fibonacci

```
i(0,0).
i(1,1).
i(N,X):-
N>1,
N1 is N-1,
fibonacci(N1,X1),
N2 is N-2,
fibonacci(N2,X2),

X is X1+X2.
```

```
lfib(N,F):- lemma_fib(N,F),!.
lfib(N,F):-
        N>1,
        N1 is N-1,
        N2 is N1-1,
        lfib(N1,F1),
        lfib(N2,F2),
        F is F1+F2,
        assert(lemma_fib(N,F)).

:-dynamic lemma_fib/2.

lemma_fib(0,0).
lemma_fib(1,1).
```

icación Dinámica: Ejercicio

rama que permita a un usuario preguntar (vía do) por la capital de un determinado país

l país está en la base de conocimientos, entonces se devuelve el nbre de su capital

l país no está en la base de conocimientos, entonces se solicita nombre de la capital y se introduce este hecho en la base de ocimientos

l usuario teclea "stop.", entonces se graba la nueva base de ocimientos y se sale del programa

itales.pl

icación Dinámica: clause/2 (I)

e(*Head*, *Body*):

ca una cláusula cuya cabeza es *Head* y cuyo cuerpo es *Body*

láusula *Head*:-*Body* existe en el programa actual

ead es un término que no es una variable libre

redicado correspondiente debe ser dinámico

plo:

) :- q(X).

) :- r(b), s(b).

).

lause(p(a), Cuerpo).

uerpo = q(a)

lause(p(b), Cuerpo).

uerpo = q(b);

uerpo = r(b), s(b)

lause(q(a), Cuerpo).

uerpo = true

icación Dinámica: clause/2 (II)

plo: Meta-intérprete simple (“*vanilla*”)

```
ve(true).
```
La meta vacía es cierta.
La meta vacía está resuelta.

```
ve((A,B)) :- solve(A), solve(B).
```
La meta conjuntiva (A, B) es cierta si A es cierta y B es cierta.
Para resolver la meta (A, B) resolver primero A y después B.

```
ve(A) :- clause(A,B), solve(B).
```
La meta A es cierta si existe una cláusula A:-B y B es cierta.
Para resolver la meta A, seleccionar una cláusula cuya cabeza unifique con A y resolver el cuerpo.

código se puede mejorar para realizar diferentes
s: *tracing*, *debugging*, proporcionar explicaciones
mas expertos), etc.

use.pl

icación Dinámica: clause/2 (III)

plo: Definir un metaintérprete que cuente la dad de hechos visitados a lo largo de la resolución na cierta consulta

```
ve(true, 1).
ve( (A, B), CHAB ) :-
olve(A, CHA), solve(B, CHB),
HAB is CHA + CHB.
ve(A, CH) :-
lause(A, B), solve(B, CH).
```

use.pl

sis Sintáctico (*Parsing*) en Prolog

ngamos que necesitamos definir un predicado que apaz de aceptar frases sencillas como:

- hermano es el hijo de tus padres
- abuelo es el padre de tus padres
- primo es el hijo de mis tios

sis Sintáctico en Prolog

frases se ajustan a la siguiente gramática BNF:

ıse>::= <sn> <sv>

>::= <posesivo> <nombre> | <determinante> <nombre>

>::= <verbo> <atributo>

ributo>::= <sn> <cn>

sesivo>::= tu | mi | mis | tus

mbre>::= hermano | abuelo | primo | padres | tios | hijo | re

rbo>::= es

terminante>::= el

>::= <prep> <sn>

ep>::= de

sis Sintáctico en Prolog

bir un predicado frase/1 que

pte una frase si se adecúa a las reglas de la gramática, o

echace en caso contrario

se se representa como una lista de palabras

rase([mi,primo,es,el,hijo,de,mis,tios]).

manera de implementar estas reglas gramaticales es

ear la estrategia de "generar y testear"

```
e(L):-
ppend(SN,SV,L),
n(SN),
v(SV).
```

Genera posibles valores para SN y SV, dividiendo la lista original L

Comprueban si cada sublista es gramaticalmente correcta. Si no, por *backtracking* se genera otra posible división

lisisSintactico-Listas.pl

Nota: Los predicados sn/1 y sv/1 son similares a frase/1, y llaman a otros predicados que tratan con unidades más pequeñas de una sentencia

g (usando *append* y listas): Ejemplo

ase, en inglés, se representa como una lista de teres

```
phrase(X) :-
ppend(A,T1,X), article(A), append(SP,T2,T1), spaces(SP),
ppend(N,T3,T2), noun(N), append(SPN,V,T3), spaces(SPN),
erb(V).
cle([a]).
cle([t,h,e]).
ces([' ']).
ces([' ' | Y]) :- spaces(Y).
n([c,a,r]).
n([p,l,a,n,e]).
b([f,l,i,e,s]).
b([d,r,i,v,e,s]).
nyphrase([t,h,e,' ',p,l,a,n,e,' ',f,l,i,e,s]).
```

EjemploParsing.pl

sis Sintáctico en Prolog

trategia “generar y testear” es ineficaz

strategia más eficiente consiste en

ar la etapa de generación

ar la lista completa a los predicados que implementan las reglas maticales

stos predicados identifican los correspondientes elementos gramaticales rocesando secuencialmente los elementos de la lista de izquierda a erecha, devolviendo el resto de la lista

ello podemos emplear listas diferencia

alisisSintactico-ListasDiferencia.pl

Diferencia (*Difference Lists*)

estructuras de datos incompletas

plo: Supongamos la lista [1, 2, 3]. Podríamos esentar dicha lista como la diferencia de los entes pares de listas:

2, 3, 5, 8] y [5, 8]

2, 3, 6, 7, 8, 9] y [6, 7, 8, 9]

,3] y [ ].

a uno de estos son casos del par de dos listas incompletas [1,2,3 y X.

ar se llama lista diferencia

ista diferencia se representa mediante A-B

una lista abierta que acaba en B

una variable libre

Diferencia (*Difference Lists*)

plo: La lista [1,2,3] se representa usando listas
encia como

2, 3 | X] – X

2, 3 | X], X) (lista abierta, referencia al resto de la lista)

iten mantener un puntero al final de la lista

iten concatenación en tiempo constante

end_dl (X-Y, Y-Z, X-Z).

iten manipular listas de forma más eficiente
iendo “patrones de listas”

Diferencia (*Difference Lists*)

cado para transformar una lista diferencia en una "normal"

```
to_list([ ] - _, [ ]) :- !.
to_list([X|Y] - Z, [X|W]) :- dl_to_list(Y - Z, W).
```

cado para transformar una lista "normal" en una diferencia

```
_to_dl([], X - X).
_to_dl([X|W], [X|Y] - Z) :- list_to_dl(W, Y - Z).
```

ncia.pl

ática Definida por Cláusulas

g incorpora la posibilidad de definir gramáticas ante una sintaxis especial que oculta la presencia de tas diferencia

sintaxis se denomina Gramática Definida por ulas (*Definite Clause Grammar* - DCG)

ensión sintáctica de la sintáxis ordinaria de Prolog

utiliza para la creación de gramáticas formales en forma eviada

plifica y hace más legibles los analizadores sintácticos

ción:

terminal --> cuerpo

o terminales: átomos de Prolog

uerpo: terminales y no terminales separados por “,”

adenas de terminales: listas de átomos de Prolog

ática Definida por Cláusulas

plos:

> [a],[b],S.

> [c].

-> sn,sv.

-> det,nom.

-> verbo,sn.

--> [el].

n --> [Pepe]

n --> [gato].

o --> [come].

plo:

cion(S0,S):- sintagma_nominal(S0,S1), sintagma_verbal(S1,S).

cion --> sintagma_nominal, sintagma_verbal. % Sintaxis DCG

ática Definida por Cláusulas

láusulas gramaticales definidas con DCG se analizan
ducen a cláusulas Prolog que usan listas diferencias

plo:

```
tence --> nounphrase, verbphrase.  % usando DCG
tence (S1, S2) : - nounphrase (S1, S3) , verbphrase (S3, S2).  % traducido
```

a: Para analizar una frase, hemos de invocar sentence/2

```
- sentence ([dog, chases, cat],R).
```

ocabulario (los símbolos terminales) en DCG se
senta con listas simples

```
oun --> [dog].
erb --> [chases].
```

s listas se traducen a listas diferencia en Prolog

```
oun([dog|X], X).
erb([chases|X], X).
```

lisisSintactico-DCGs.pl

ática Definida por Cláusulas

ramática definida presenta algunas deficiencias
o la falta de concordancia entre el número del
sivo y el nombre

ejemplo, “mi padres” resultaría aceptable como
gma nominal (sn):

n([mi,padres], R).
= []
es

llo una frase como la siguiente sería aceptable:

rase([mi,abuelo,es,el,padres,de,mis,padre],R).
= []
es

tica Definida por Cláusulas: Uso de Variables

resolver este problema se pueden emplear mentos en las cláusulas de la gramática

```
esivo(sing) --> [mi].
esivo(plur) --> [mis].
nbre(plur) --> [padres].
nbre(sing) --> [padre].
```

esta solución la siguiente frase no es válida

```
rase([mi,primo,es,el,hijo,de,mi,tios],R).
```

o

lisisSintactico-DCGs-SingPlu.pl

tica Definida por Cláusulas: Uso de Variables

plo: Uso de variables para devolver un análisis ctico (y eventualmente morfológico) de la frase

alisisSintactico-DCGs-Analisis.pl

ática Definida por Cláusulas: Acciones

osible incluir cláusulas de Prolog en la definición de áusulas gramaticales.

cláusulas deben encerrarse entre llaves { }

plo:

```
?- myphrase(NChars,"the plane flies",[]).
?- phrase(myphrase(N),"the plane flies").
se_package(dcg).
hrase(N) --> article(AC), spaces(S1), noun(NC), spaces(S2),
erb(VC), { N is AC + S1 + NC + S2 + VC}.
cle(3) --> "the".          spaces(1) --> " ".
cle(1) --> "a".            spaces(N) --> " ", spaces(N1), {N is N1+1}.
n(5) --> "plane".          verb(5) --> "flies".
n(3) --> "car".            verb(6) --> "drives".
```

: Ejemplo

ática para reconocer expresiones aritméticas.

```
--> term.
--> term, [+], expr.
--> term, [-], expr.
--> num.
--> num, [*], term.
--> num, [/], term.
--> [D], { number(D) }.
(E) :- expr(E,[]).
```

lisisSintactico-ExpresionesAritmeticas.pl

: Ejercicio

bir una gramática DCG para analizar contactos
ónicos

to_contacto('Garcia, Manolo 992168010').
to_contacto('Pedro 634873429').
to_contacto('Perea, Jose Manuel 634987450').
to_contacto('Lopez Carmona, Carmen 594270328').
to_contacto('Castro Moreno, Luis Miguel 340769600').
to_contacto('Martinez de la Rosa, Pedro 917349238').

contactos.pl

Programación Declarativa: Lógica y Restricciones

l lenguaje de programación ISO-Prolog

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