Programación Declarativa:
Lógica y Restricciones

El 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 (I)

inos Prolog:

constante

- átomo
- número
  - entero
  - real

variable

estructura

cados para tipos:

Son relaciones unarias que permiten comprobar el tipo de un termino

Tienen éxito o fallan, pero no producen error

No se pueden usar para generar

- Si el argumento es una variable, fallan en lugar de instanciarla a valores posibles

ados para Tipos (II)

plos de Predicados para Tipos:

nteger(X)

oat(X)

umber(X)

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

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

ompound(X) → X es una estructura

ados para Tipos (III): Ejemplos

```
om(vacio).
Yes
eger(-3).
Yes
mpound([a,b|Xs]).
Yes
mpound(-3.14).
No
eger([1]).
No
: 2, integer(X).
Yes
mpound([X]).
Yes
```

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)

cados 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

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

Y < X+1, X is Y+1, X =:= Y. → fallo

Y < a+1, X is Z+1, X =:= f(a). → error

ética (III)

plos:

?- 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)

X is 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

Sólo funciona en modo (in, in, out)

X e Y deben ser términos aritméticos

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
```

cado 'suma' entre enteros para cubrir el caso en el
os sumandos puedan no estar instanciados pero el
tado 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

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

cada término, salvo los dos primeros, es la suma de los dos anteriores

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

?- fibonacci(6,X).

- X = 8

ética: Ejercicio 2

ir lista_numeros/3 (lista_numeros(N,M,L)) que se ca si L es la lista de los números entre N y M, ambos sive

?- lista_numeros(3,5,L).

- L = [3,4,5]

?- lista_numeros(3,2,L).

- no

ética: Ejercicio 3

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

?- mcd(10,15,X).

- X=5

Solución: Usar el **algoritmo de Euclides**: (con restas)

- Si los dos números son iguales, el mcd es cualquiera de ellos
- Si no, al mayor se le resta el menor y se comprueba si la diferencia es igual al menor
- Si no, se repite la operación

o a Estructuras

meta-predicados de **inspección de estructuras** iten:

Descomponer una estructura en sus componentes

Componer una estructura a partir de sus componentes

g proporciona 3 meta-predicados de inspección:

functor/3

arg/3

=../2

o a Estructuras: functor/3

or(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)

plos:

?- 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

modo (in, in, in) 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

l modo (in, out, out) se utiliza para obtener el or principal de un término

?- functor(punto(a,X),F,N).

- F = punto
- N = 2

?- functor([A,f(X),Y],F,N).

- F = '.'
- N = 2

o a Estructuras: functor/3

modo (out, in , in) se comporta como un generador

: 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

o a Estructuras: arg/3

, E, C): la estructura E tiene el componente C en la ión 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

plos:

?- _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 modo (in, in, out) se utiliza para obtener el nento 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).

- A = [b, c, d]

?- arg(3,[a,b,c,d],A).

- % fuera de rango
- no

o a Estructuras: arg/3

l modo (in, out, in) se utiliza para instanciar el nento i-ésimo 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

```
subTerm(X,X).
subTerm(X,Y):-
        compound(Y),
        functor(Y,F,N),
        subTerm(N,X,Y).

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

Cualquier término es subtérmino de si mismo

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

subTerm/3 comprueba iterativamente todos los argumentos

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

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 los
:ros 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

Los registros se recorren desde el final hasta el principio (para usar un solo índice, deteniéndose a 0), y se suman los elementos correspondientes

o a Estructuras: =../2

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]

rta los modos (in, in), (out, in) e (in, out)

odo (out, out) genera un error

?- A =.. B.

- instantiation error

o a Estructuras: =../2

modo (in, in) 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

o a Estructuras: =../2

l modo (in, out) se utiliza para descomponer un ino 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 = [[]]

o a Estructuras: =../2

modo (out, in) se comporta como un generador
. Se utiliza para componer un término a partir de
omponentes

?- 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 1

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

cuadrado(lado); rectangulo(anchura, altura); triangulo(lado1, lado2, 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)

o a Estructuras: =../2. Ejercicio 1

```
la(Fig,K,KFig) :-
ig =.. [Tipo|Ds],       % descomponer figura
ultiplica(K,Ds,KDs),
Fig =.. [Tipo|KDs].  % componer figura

tiplica(_,[],[]).
tiplica(K,[D|Ds],[KD|KDs]) :-
D is K*D,
ultiplica(K,Ds,KDs).
```

o a Estructuras: =../2. Ejercicio 2

ir el predicado subterm(X,Y) usando "=../2"

```
ubterm(Term,Term).
ubterm(Sub,Term) :-
        compound(Term),
        Term =.. [F|Args],
        subtermList(Sub, Args).

ubtermList(Sub,[Arg|Args]) :-
        subterm(Sub,Arg).
ubtermList(Sub,[Arg|Args]) :-
        subtermList(Sub,Args).
```

cio 1: Acceso a Estructuras

oniendo que un polígono se representa por su
ıbre y las longitudes de sus lados.

nir el predicado **es_equilátero/1** que se verifica si el
gono P es equilátero (es decir, que todos sus lados
iguales)

nplos:

s_equilatero(triangulo(4,4,4)).

yes

s_equilatero(cuadrilatero(3,4,5,3)).

No

cio 2: Acceso a Estructuras

onemos grafos representados mediante listas de as coloreadas, que están representadas por una uctura de 3 argumentos, de la que sólo se sabe que rcer argumento es el color.

pide definir un predicado **colorear_grafo/3** orear_grafo(Grafo,Color,GrafoColoreado)) que tiña afo del color indicado.

```
colorear_grafo([],_,[]).
colorear_grafo([X|Xs],C,[Y|Ys]):-
        X =.. [N,A,B,_],
        Y =.. [N,A,B,C],
        colorear_grafo(Xs,C,Ys).
```

ados Meta-Lógicos

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 está totalmente instanciada

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

aración de Términos

gualdad de términos puede determinarse de entes formas

perador "=" es la propia unificación

e unifican las variables de los términos que se comparan

perador "==" (idéntico) no unifica las variables de los términos se comparan

or tanto, una variable (no ligada) sólo será igual a sí misma

mplos:

1 = T2	Es cierto si T1 y T2 pueden unificarse
1 \= T2	Es cierto si T1 y T2 no pueden unificarse
1 == T2	Es cierto si T1 y T2 son idénticos
1 \== T2	Es cierto si T1 y T2 no son idénticos

aración de Términos: Ejemplos

:= a.

:= X.

:= Y.

:= X.

= f(X).

) == f(Y).

ación de Términos No Básicos

n alfabético/lexicográfico:

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

ejemplo: T1 @< T2 se verifica si el término T1 es anterior a T2
el orden 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)

pueden 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

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).
```

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), write(' es '), write(C), nl,
ubo.
```

la/Salida de Caracteres

cado get_code(X):

el primer carácter imprimible y unifica su código ASCII con X

mplo:

- get_code(X).

: d

= 100

cado put_code(X):

ribe el carácter correspondiente al código ASCII instanciado en X

mplo:

- put_code (104).

la/Salida de Ficheros (I)

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

ome/usuario/fichero.txt'

da.txt'

cado see(X):

ablece como canal de entrada el fichero X

no está instanciada, el predicado falla

cado seeing(X):

rigua el canal de entrada activo

cado seen:

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

la/Salida de Ficheros (II)

cado tell(X):

ablece como canal de salida el fichero X

no está instanciada, el predicado falla

cado telling(X):

rigua el canal de salida activo

cado told:

ra el fichero y restablece el teclado como canal de salida

la/Salida: Ejercicio

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

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

*
****
**
***
es

ados de Orden Superior (I)

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

ados de Orden Superior (II)

n predicados de orden superior:

ellos que permiten añadir o eliminar cláusulas del conjunto soporte
grama) en tiempo de ejecución (programación dinámica)

assert/1, retract/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

debe estar instanciado a un término, sino se produce un error (de
stanciación)

) se cumple si se satisface X como objetivo

a habitualmente para

ta-programación (intérpretes, *shells*)

inir negación

lementar orden superior

plo:

).

) :- call(X).

(q(Y)).

a

Programación: call/1. Ejemplos

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(_).

ado fail/0

redicado fail/0 es un predicado predefinido que
ore falla

mpre produce fallo

a cuando se ejecuta

milar 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

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 se representa mediante el cado predefinido de segundo 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 1

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

ultas

probado(luis).

es

probado(X).

o

ogramación: Negación como Fallo. Ejemplo 1

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

¿Qué pasa si cambiamos el orden de los predicados?

?- aprobado2 (X).

ogramación: Negación como Fallo. Ejemplo 2

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
```

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

ución 1: definición con not

ución 2: definición con corte

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

cio

ión 1: Definición con not

```
ra_1([],_,[]).
ra_1([X|L1],Y,L2) :-
        X=Y,
        borra_1(L1,Y,L2).
ra_1([X|L1],Y,[X|L2]) :-
        not(X=Y),
        borra_1(L1,Y,L2).
```

cio

ión 2: Definición con corte

```
ra_2([],_,[]).
ra_2([X|L1],Y,L2) :-
        X=Y, !,
        borra_2(L1,Y,L2).
ra_2([X|L1],Y,[X|L2]) :-
        borra_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 de 2º Orden

do meta-predicados podemos definir predicados de
ndo orden:

decir, predicados que razonan sobre conjuntos/relaciones
re objetos y no sobre los objetos mismos (1er orden)

mplos:

efinir el conjunto de todas las soluciones de un objetivo

efinir la igualdad entre listas al modo de conjuntos

efinir el cierre transitivo de una relación

efinir una relación de equivalencia a partir de otra relación dada

efinir la aplicación de una operación sobre los elementos de una lista

efinir el conjunto de todas las soluciones a un objetivo y que cumplen
na condición ulterior

efinir el cumplimiento de una condición por parte de todos los
lementos de una lista

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

número 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 propio de un número natural N, a número también natural que es divisor de N, pero ente de N

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

ir el predicado factoresPropios/2 oresPropios(+N,-L)) que se verifique si L es la lista nada de los factores propios del número N

actoresPropios(42,L).

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

cio 2

úmeros naturales se pueden clasificar en tres tipos:

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

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

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

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

Natural(10,T).
=a

Natural(28,T).
=b

Natural(12,T).
=c

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

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

cio 5

ir el predicado frecuentes/2 (frecuentes(+L1,-L2))
se verifica si L2 es la lista de los elementos de L1 que
den el mayor número de veces

recuentes([1,2,1,2,3,3,4,5,5,0],L).

= [1,2,3,5]

cio 6

ir el predicado cartesiano/3 (cartesiano(L1,L2,L3))
enga éxito si L3 es el producto cartesiano de L1 y L2

artesiano([a,b],[1,2,3],C).

= [(a,1),(a,2),(a,3),(b,1),(b,2),(b,3)]

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 redundantes

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

comprueba si Clausula existe en la base de conocimientos

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).
```

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).
```

Las llamadas '*memo-functions*' guardan resultados de cálculos intermedios para usarlos posteriormente.
Los predicados 'lemma' son típicamente '*memo-functions*' que evitan recalcular ciertos resultados.

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 muestra 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*”) (intérprete
n lenguaje escrito en el propio lenguaje)

```
ve(true).

ve((A,B)) :- solve(A), solve(B).

ve(A) :- clause(A,B), solve(B).
```

arativa:
- ía es cierta
- juntiva (A, B) es cierta si A es cierta y B es cierta
- s cierta si existe una cláusula A:-B y B es cierta

Lectura Procedimental:
- La meta vacía está resuelta
- Para resolver la meta (A, B) resolver primero A y después B
- 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.

icación Dinámica: clause/2 (III)

plo: Definir un meta-intérprete 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).
```

sis Sintáctico (*Parsing*) en Prolog (I)

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 (II)

frases se ajustan a la siguiente gramática BNF:

ase>::= <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 (III)

bir un predicado frase/1 que

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

echace en caso contrario

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

n(SN),

v(SV). 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

sis Sintáctico en Prolog (IV)

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*) (I)

estructuras de datos incompletas

plo: La lista [1, 2, 3] se podría representar como la encia de los siguientes 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 ejemplos son casos del par de dos listas ompletas [1,2,3 | X] y X

ar se llama **lista diferencia**

istas diferencia siempre llevan una variable como de lista

na variable debe guardarse como segundo elemento del par

ista diferencia se representa mediante A-B o A\B

s una lista abierta que acaba en B ([a,b,c|B])

s una variable libre

Diferencia (*Difference Lists*) (II)

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*) (III)

cado para transformar una lista diferencia en una "normal"

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


l_to_list([1,2,3|X]-X,L).
[ ]
[1,2,3];
```

Diferencia (*Difference Lists*) (IV)

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).


st_to_dl([a,b,c],Y-Z).
[a, b, c|_G167]
_G167 ;
```

ática Definida por Cláusulas (I)

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 (II)

plos:

> [a],[b],S.

> [c].

-> sn,sv.

-> det,nom.

-> verbo,sn.

--> [el].

n --> [perro].

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 (III)

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 (IV)

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

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",[]).
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,[]).
```

: 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').

Programación Declarativa:
Lógica y Restricciones

El Lenguaje de Programación ISO-Prolog

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