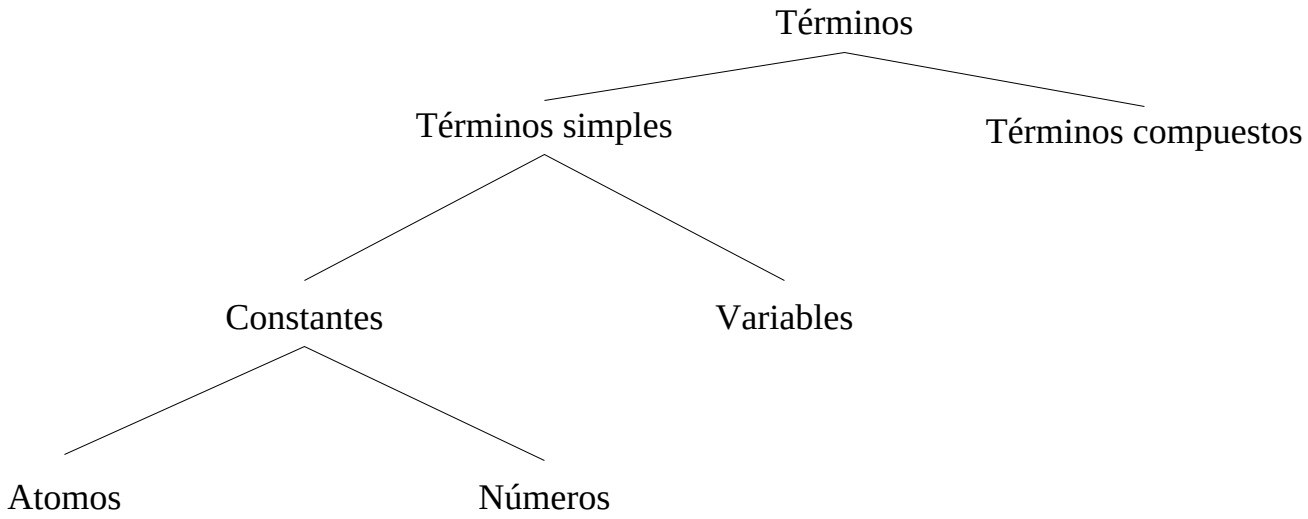
Tema 5: Otros predicados predefinidos

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Clasificación de términos



- **Números**
  - **Número:** `number`

```
?- number(5.468).
Yes
?- number(-4).
Yes
?- number(a).
No
```

  - **Enteros:** `integer`

```
?- integer(4).
Yes
?- integer(2.345).
No
```

Clasificación de términos

- **Punto flotante:** `float`

```
?- float(3.702).
Yes
?- float(-7).
No
```

- `round(+Expr)` **evalúa** `Expr` **y redondea el resultado al entero más próximo**

```
?- X is round(5.3).
X = 5
Yes
?- X is round(5.8).
X = 6
Yes
?- X is round(127/10).
X = 13
Yes
?- X is round((127/10)*1.34).
X = 17
Yes
?- X is round(5.5).
X = 6
Yes
?- X is round(5.4999).
X = 5
Yes
```

Clasificación de términos

- **Suma segura**
  - `suma_segura(X,Y,Z)` **se verifica si** `X` **e** `Y` **son enteros y** `Z` **es la suma de** `X` **e** `Y`
  - **Ejemplo:**

```
?- suma_segura(2,3,X).
X = 5
Yes

?- suma_segura(7,a,X).
No

?- X is 7 + a.
[WARNING: Arithmetic: 'a' is not a function]
```

  - **Programa** `suma_segura.pl`

```
suma_segura(X,Y,Z) :-
   integer(X),
   integer(Y),
   Z is X+Y.
```

Clasificación de términos

- **Átomos y variables:** `atom`, `atomic`, `var` y `nonvar`
  - `atom(+T)` **se verifica si** `T` **está ligada a un átomo**
  - `atomic(+T)` **se verifica si** `T` **está ligada a un átomo, una cadena o un número entero o flotante**
  - **Ejemplos**

```
?- atom(ana).
Yes
?- atom('Ana').
Yes
?- atom(Ana).
No
?- atom(5.789).
No
?- atomic(5.789).
Yes
?- var(X).
X = _G105
Yes
?- X=a,var(X).
No
?- nonvar(a).
Yes
```

Clasificación de términos

- **Cuenta ocurrencias**
  - `cuenta(A,L,N)` **se verifica si** `N` **es el número de ocurrencias del átomo** `A` **en la lista** `L`**.**
  - **Ejemplos:**

```
?- cuenta(a,[a,b,a,a],N).
N = 3
Yes
?- cuenta(a,[a,b,X,Y],N).
X = _G313
Y = _G316
N = 1
Yes
```

  - **Programa** `cuenta-1.pl`

```
cuenta(_,[],0).
cuenta(A,[B|L],N) :-
   atom(B), A=B, !,
   cuenta(A,L,M),
   N is M+1.
cuenta(A,[B|L],N) :-
   % not(atom(B), A=B)
   cuenta(A,L,N).
```

Clasificación de términos

- **Si se cambia la definición anterior suprimiendo el literal `atom(B)`...**

```
cuenta_1(_,[],0).
cuenta_1(A,[B|L],N) :-
   A=B, !,
   cuenta_1(A,L,M),
   N is M+1.
cuenta_1(A,[B|L],N) :-
   % not(A=B)
   cuenta_1(A,L,N).
```

- **... entonces, se obtienen los siguientes resultados**

```
?- cuenta_1(a,[a,b,a,a],N).
N = 3
Yes

?- cuenta_1(a,[a,b,X,Y],N).
X = a
Y = a
N = 3 ;
No
```

Manipulación de términos

- **El predicado =..**
  - **`?T =.. ?L` se verifica si `L` es una lista cuya primer elemento es el functor del término `T` y los restantes elementos de `L` son los argumentos de `T`**
  - **Ejemplos**

```
?- padre(juan,luis) =.. L.
L = [padre, juan, luis]

?- T =.. [padre, juan, luis].
T = padre(juan,luis)
```

- **Figuras proporcionales**
  - **Las figuras geométricas se representan como términos en los que el functor indica el tipo de figura y los argumentos su tamaño. Por ejemplo,**

```
cuadrado(3)
triangulo(3,4,5)
circulo(2)
```

  - **`alarga(Figura1,Factor,Figura2)` se verifica si `Figura1` y `Figura2` son figuras geométricas del mismo tipo y el tamaño de la `Figura2` es el de la `Figura1` multiplicado por `Factor`.**

Manipulación de términos

- **Ejemplo**

```
?- alarga(triangulo(3,4,5),2,F).
F = triangulo(6, 8, 10)

?- alarga(cuadrado(3),2,F).
F = cuadrado(6)
```

- **Programa** `alarga.pl`

```
alarga(Figura1,Factor,Figura2) :-
   Figura1 =.. [Tipo|Argumentos1],
   multiplica_lista(Argumentos1,Factor,Argumentos2),
   Figura2 =.. [Tipo|Argumentos2].

multiplica_lista([],_,[]).
multiplica_lista([X1|L1],F,[X2|L2]) :-
   X2 is X1*F,
   multiplica_lista(L1,F,L2).
```

• Sustitución de un término

- `sustituye(Sub1,Term1,Sub2,Term2)` **se verifica si** `Term2` **es el término obtenido sustituyendo todas las ocurrencias de** `Sub1` **en** `Term1` **por** `Sub2`**.**

Manipulación de términos

- **Ejemplo**

```
?- sustituye(sen(x),2*sen(x)*f(sen(x)),y,T).
T = 2 * y * f(y)
Yes

?- sustituye(a+b,f(a,A+B),c,T).
A = a
B = b
T = f(a, c)
Yes
```

- **Programa** `sustituye.pl`

```
sustituye(Sub1,Term1,Sub2,Sub2) :-
   Sub1 = Term1, !.
sustituye(Sub1,Term1,Sub2,Term1) :-
   % not(Sub1 = Term1),
   atomic(Term1), !.
sustituye(Sub1,Term1,Sub2,Term2) :-
   % not(Sub1 = Term1),
   % not(atomic(Term1)),
   Term1 =.. [Functor|Args1],
   sustituye_lista(Sub1,Args1,Sub2,Args2),
   Term2 =.. [Functor|Args2].

sustituye_lista(_,[],_,[]).
sustituye_lista(Sub1,[T1|T1s],Sub2,[T2|T2s]) :-
   sustituye(Sub1,T1,Sub2,T2),
   sustituye_lista(Sub1,T1s,Sub2,T2s).
```

Manipulación de términos

- **Términos cerrados**
  - `cerrado(T)` **se verifica si** `T` **es un término cerrado (es decir, sin variables).**
  - **Ejemplo**

```
?- cerrado(f(a+b)).
Yes
?- cerrado(f(a+X)).
No
```

  - **Programa** `cerrado.pl`

```
cerrado(T) :-
  nonvar(T),
  T =.. [_|Args],
  cerrados(Args).

cerrados([]).
cerrados([X|L]) :-
  cerrado(X),
  cerrados(L).
```

  - **El predicado predefinido** `ground` **tiene el mismo efecto que** `cerrado`**.**

Manipulación de términos

- **El predicado `maplist`**
  - **`n_maplist(+Predicado, ?Lista1, ?Lista2)` se verifica si se verifica el `Predicado` sobre los sucesivos pares de elementos de la `Lista1` y la `Lista2`**
  - **Ejemplos**

```
?- n_maplist(succ,[2,4],[3,5]).
Yes
?- n_maplist(succ,[2,4],Y).
Y = [3, 5]
Yes
?- n_maplist(succ,X,[3,5]).
X = [2, 4]
Yes
?- n_maplist(succ,[0,4],[3,5]).
No
```

  - **Programa `n_maplist.pl`**

```
n_maplist(_,[],[]).
n_maplist(F,[X1|L1],[X2|L2]) :-
   G =.. [F,X1,X2],
   G,
   n_maplist(F,L1,L2).
```

  - **El predicado predefinido `maplist` tiene el mismo efecto que `n_maplist`.**

Manipulación de términos

- **Los predicados `functor` y `args`**
  - **Ejemplos**

```
?- functor(f(a,b,c),SF,Aridad).
SF = f
Aridad = 3 ;
No

?- Term =..[padre,juan,luis],functor(Term,SF,Aridad).
Term = padre(juan, luis)
SF = padre
Aridad = 2
Yes

?- Term =..[padre,juan,luis],arg(N,Term,Argumento).
Term = padre(juan, luis)
N = 1
Argumento = juan ;
Term = padre(juan, luis)
N = 2
Argumento = luis ;
No
```

- **El predicado `name`**

```
?- name(finidi,L).
L = [102, 105, 110, 105, 100, 105]
Yes
?- name(L,[116, 115, 97, 114, 116, 97, 115]).
L = tsartas
Yes
```

Manipulación de términos

- **Números de Turing**
  - **Un número entero es de Turing si es igual al cuadrado de la suma de sus cifras. Definir la relación `numero_de_turing(N)` que se verifique si `N` es un número de Turing y calcular los números de Turing menores que 1.000.**
  - **Programa `turing.pl`**

```
numero_de_turing(N) :-
   cifras(N,L),
   suma_cifras(L,M),
   N =:= M**2.

cifras(N,L) :-
   name(N,L1),
   reduce(L1,L).

reduce([],[]).
reduce([X|R],[Y|S]) :-
   Y is X - 48,
   reduce(R,S).

suma_cifras([],0).
suma_cifras([X|R],N) :-
   suma_cifras(R,N1),
   N is X+N1.
```

  - **Sesión**

```
?- between(0,1000,N),numero_de_turing(N).
N = 0 ;
N = 1 ;
N = 81 ;
No
```

Varias clases de igualdad

- **Unificación =, \=**

```
?- f(a,X)=f(Y,b).
X = b
Y = a ;
No

?- [X,Y,Z] = [Y,Z,X].
X = _G159
Y = _G159
Z = _G159 ;
Nos
?- f(a,b) \= f(_X,_X).
Yes
```

- **Test de ocurrencia**
  - **Programa** `monstruo.pl`

```
monstruo :- X=f(X).
```

  - **Sesión**

```
?- monstruo.
Yes
```

Varias clases de igualdad

- **Aritmética** `is`, `=:=`, `=\=`

```
?- X is 3+2.
X = 5 ;
No

?- 5 is 3+X.
[WARNING: Arguments are not sufficiently instantiated]

?- 11 - 3 =:= 2^3.
Yes

?- 11 - 3 =:= 2^X.
[WARNING: Arguments are not sufficiently instantiated]

?-  11 - 3 =\= 2^7.
Yes

?- 11 - X =\= 2^7.
[WARNING: Arguments are not sufficiently instantiated]
```

Varias clases de igualdad

- **Identidad ==, \==**

```
?- f(a,b) == f(a,b).
Yes

?- f(a,b) == f(a,X).
No

?- f(a,Y) == f(a,X).
No

?- X \== Y.
X = _G111
Y = _G112
Yes

?- _X \== _Y.
Yes

?- f(A,g(B,i),Z) == f(A,g(B,i),Z).
A = _G240
B = _G237
Z = _G242
Yes
```

Varias clases de igualdad

• Cuenta ocurrencias (II)

- `cuenta(A,L,N)` **se verifica si** `N` **es el número de ocurrencias idénticas al átomo** `A` **en la lista** `L`**.**
- **Ejemplos**

```
?- cuenta(a,[a,b,a,a],N).
N = 3
Yes

?- cuenta(a,[a,b,X,Y],N).
X = _G313
Y = _G316
N = 1
Yes
```

- **Programa** `cuenta-2.pl`

```
cuenta(_,[],0).
cuenta(A,[B|L],N) :-
   A == B,
   cuenta(A,L,M),
   N is M+1.
cuenta(A,[B|L],N) :-
   A \== B,
   cuenta(A,L,N).
```

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 2: "Syntax and Meaning of Prolog Programs"
  - Cap. 7: "More Built–in Procedures"
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
  - Cap. 2: "A closer look"