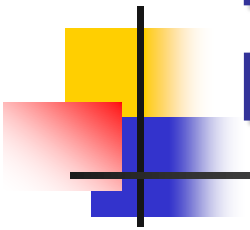
Estructuras de Datos y Listas en PROLOG

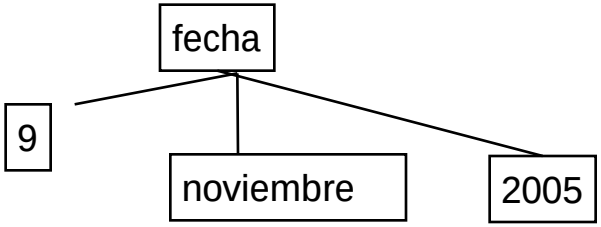


1. Estructuras y árboles
2. Listas
3. Operaciones con Listas
4. Ejercicios
5. Prácticas

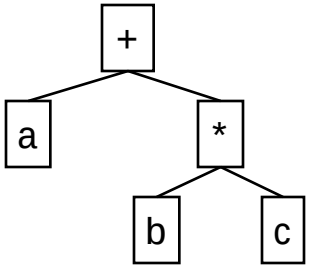
1. Estructuras y árboles

- Una estructura de datos en PROLOG se puede visualizar mediante un árbol

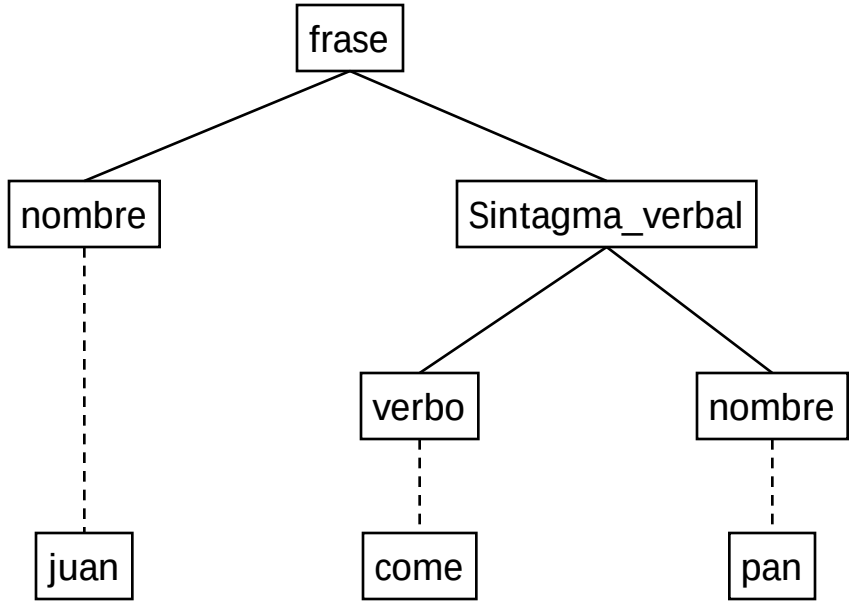
fecha(9, noviembre, 2005)



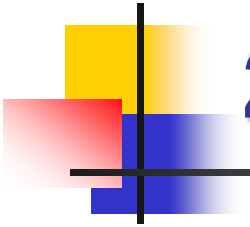
a+b*c



frase(nombre, sintagma_verbal(verbo, nombre))



frase(nombre(juan), sintagma_verbal(verbo(come), nombre(pan)))



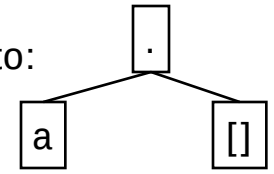
2. Listas

- Una lista es una secuencia “ordenada” de términos (constantes, variables, estructuras, e, incluso, otras listas).
- Usos de Listas: análisis sintáctico, gramáticas, diagramas de flujo, grafos, etc.
- Manejo específico de listas -> LISP.
- La lista es un caso particular de estructura en PROLOG => definición recursiva.

2. Listas

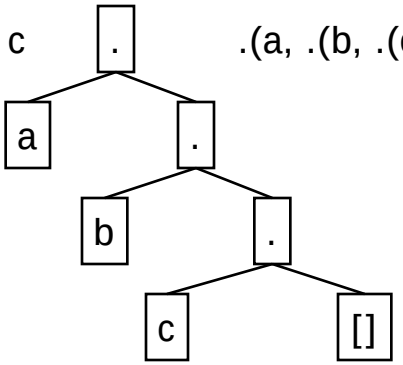
- Tipo particular de árbol: cabeza y cola
  - El functor/estructura asociado es “.”
  - El final de la lista es “[]”

Lista de un solo elemento:
.(a, [])



Lista: a, b, c

.(a, .(b, .(c,[])))



Representación habitual de las listas:

[a,b,c]

[los, hombres, [van, a, pescar]]

Cabeza es el primer término.

Cola es una lista que contiene al resto.

Un forma de instanciar a una lista con cabeza X y cola Y sería:

[X|Y]

3. Operaciones con Listas (I)

Pertenencia

- Saber si un objeto pertenece a lista.

  [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]

- Construir el predicado “miembro”:
  - miembro(X, [X|_]). (⇔ miembro(X, [Y|_]):- X=Y.)
    - sólo comprueba coincidencia con la cabeza
  - miembro(X, [_|Y]) :- miembro(X,Y).
    - Verifica la coincidencia a la cola y de forma recursiva va operando sobre otra lista progresivamente más pequeña
  - ?- miembro(felipe_ii, [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]).
  - ?- miembro(X, [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]).

Recursión (I)

- Natural en muchas definiciones declarativas
- Perspectiva operacional: natural si misma operación sobre distintos datos

- Esquema genérico

1. Caso base: entero(0).
2. Luego recurrir: entero(X):- entero(Y), X is Y+1.

Recursión (II)

Pertenencia (II)

- Cuidado con la recursión por la izquierda:

```
enteroMal(X):- enteroMal(Y), X is Y+1.
enteroMal(0).

?- enteroMal(X).
   ERROR: Out of local stack
```

Colocar la regla como última cláusula de programa

- Evitar definiciones circulares:

```
padre(X, Y) :- hijo(Y, X).
hijo(A, B) :- padre(B, A).
```

(se entra en un bucle infinito)

3. Operaciones con listas (II)

Insertar un elemento

- Queremos introducir un elemento X al comienzo de la Lista.
  - El elemento X pasará a ser la nueva cabeza de la nueva lista.
  - El cuerpo de la nueva lista será la antigua Lista.
- Definición:

```
insertar(X, Lista, Resultado) :- Resultado = [X|Lista].
```

Versión compacta:

```
insertar2(X, Lista, [X|Lista]).
```

- Ejemplos:

```
?- insertar(1, [3, 5, 7], Primos).
   Primos = [1, 3, 5, 7]
   Yes
?- insertar2(rojo, [verde, azul], Colores).
   Colores = [rojo, verde, azul]
   Yes
```

3. Operaciones con listas (III)

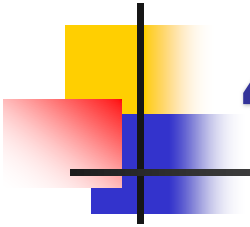
Predicado "Concatena"

- Existe un predicado predefinido *append*:

```
?- append([a, b, c], [1, 2, 3], X).
X=[a, b, c, 1, 2, 3]
?- append(X, [b, c, d], [a, b, c, d]).
X=[a]
?- append([a], [1, 2, 3], [a, 1, 2, 3]).
Yes
?- append([a], [1, 2, 3], [alfa, beta, gamma]).
No
```

- Definición:

```
concatena([], L, L).
concatena([X|L1], L2, [X|L3]) :- concatena(L1, L2, L3).
```



4. Ejercicios (I)

Definir las siguientes operaciones sobre listas:

1. *es_lista? (Lista)*
   Se corresponde Lista con la definición de lista en PROLOG.
2. *longitud(L, Num)*
   Determina el número de elementos de L.
3. *ultimo (Ult, Lista)*
   ultimo elemento de la lista.
4. *Subconjunto(Lista1, Lista2)*
   ¿Es Lista1 un subconjunto de Lista2?

4. Ejercicios (II)

Alterar frase

- Diálogo:
  - Usuario: tu eres un ordenador
  - Prolog: yo no soy un ordenador
  - Usuario: hablas francés
  - Prolog: no_hablo alemán (“_” simula a “,”)
- Reglas:
  - Aceptar frase en formato lista:
    [tu, eres, un ordenador]
  - Cambiar cada *eres* por un [*no, soy].*
  - Cambiar cada *hablas* por un [*no_,hablo].*
  - Cambiar cada *francés* por un *alemán.*

Solución: alterar frase

```
cambiar(tu, yo).
cambiar(eres, [no, soy]).
cambiar(hablas, [no_,
    hablo]).
cambiar(frances, aleman).
cambiar(X,X).


alterar([], []).
alterar([H|T], [X|Y]):-
          cambiar(H,X),
          alterar(T,Y).
```

```
?- alterar([tu, hablas, frances], X).

X = [yo, [no_, hablo], aleman] ;

X = [yo, [no_, hablo], frances] ;
X = [yo, hablas, aleman] ;
X = [yo, hablas, frances] ;
X = [tu, [no_, hablo], aleman] ;
X = [tu, [no_, hablo], frances] ;
X = [tu, hablas, aleman] ;
X = [tu, hablas, frances] ;
```

4. Ejercicios (III)

Ordenación alfabética (I)

- Las palabras se considerarán una lista de números (enteros) correspondiente a su secuencia ASCII.

- Obtener la lista asociada a cada palabra:

```
?- name(pala, X).
X=[112, 97, 108, 97].
```

- Comparación:

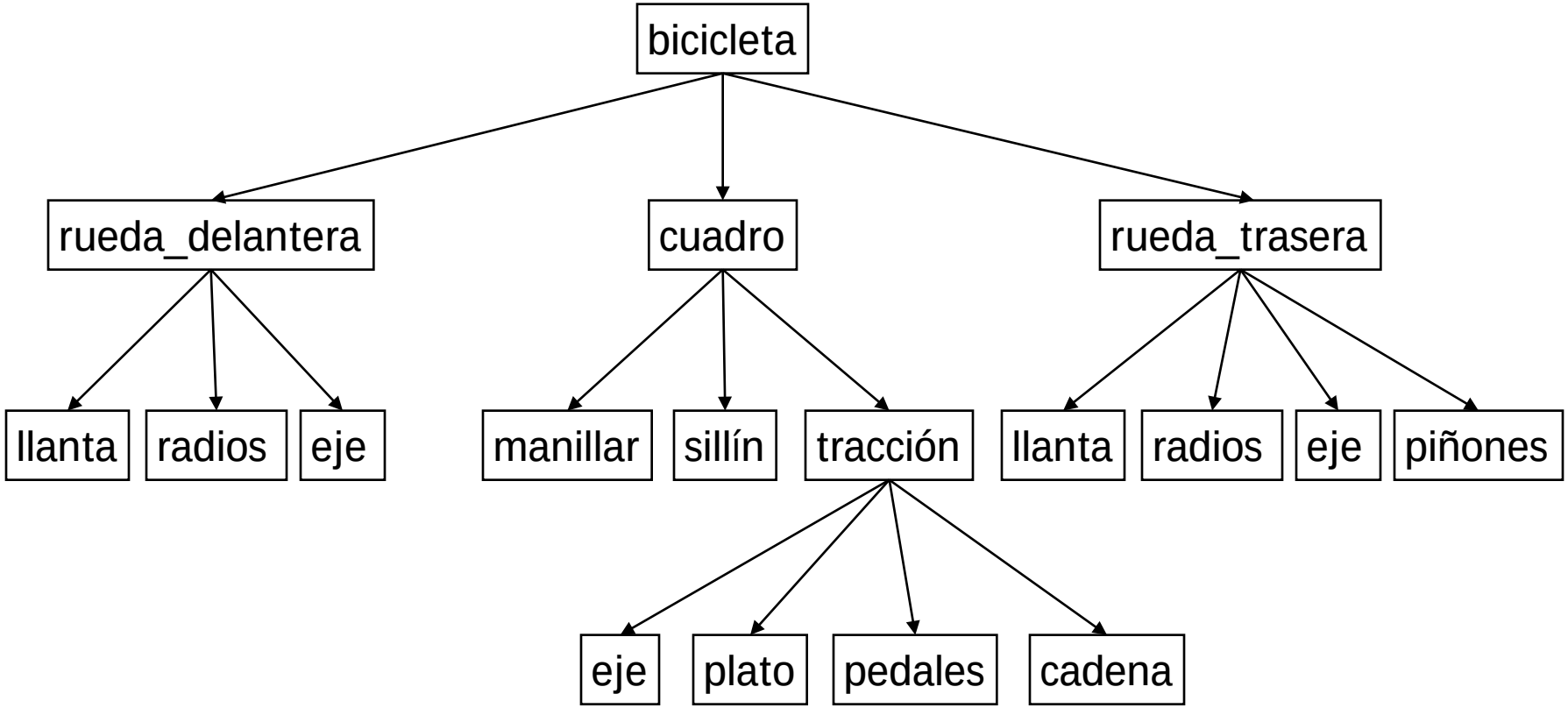
```
amenor(X, Y) :- name(X, L), name(Y, M), amenorx(L, M).
amenorx([], [_|_]).
amenorx([X|_], [Y|_]) :- X<Y.
amenorx([A|X], [B|Y]) :- A=B, amenorx(X, Y).
```

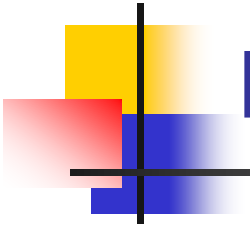
Ordenación alfabética (II)

- Ejercicios:
  - ¿qué sucede si se elimina la relación amenor([], [_|_])?
  - Modifíquese el fichero PROLOG para admitir dentro de “amenor” el caso de dos palabras iguales.

5. Prácticas

Inventario de piezas (I)





Práctica: inventario de piezas (II)

- Definir el árbol mediante las relaciones:
  - pieza_basica(cadena).
  - ensamblaje(bicicleta, [rueda_delantera, cuadro, rueda_trasera]).
- Construir relaciones “piezas_de”, que sirva para obtener la lista de piezas básicas para construir una determinada parte de (o toda) la bicicleta.