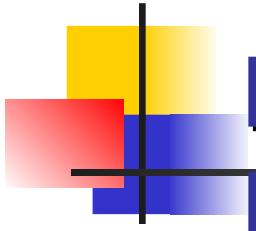


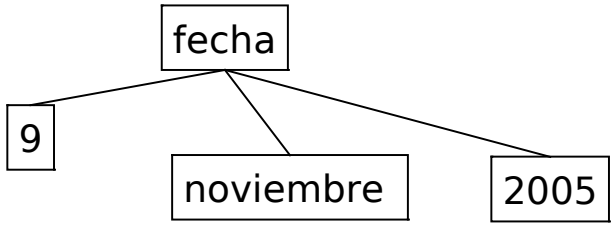
Estructuras de Datos y Listas en PROLOG

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

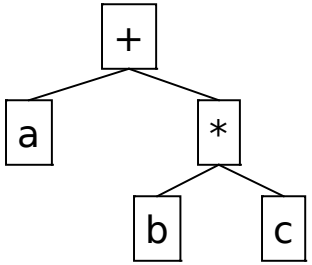
1. Estructuras y árboles

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

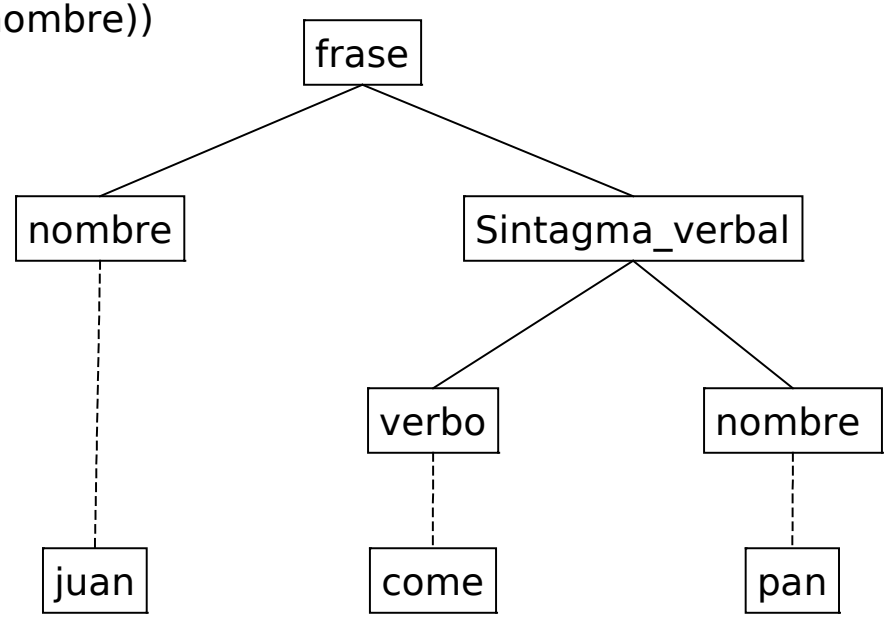
fecha(9, noviembre, 2005)



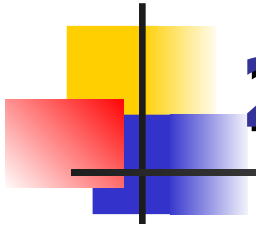
a+b*c



frase(nombre, sintagma_verbal(verbo, nombre))



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



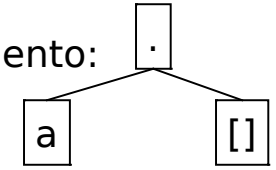
2. Listas

- 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.

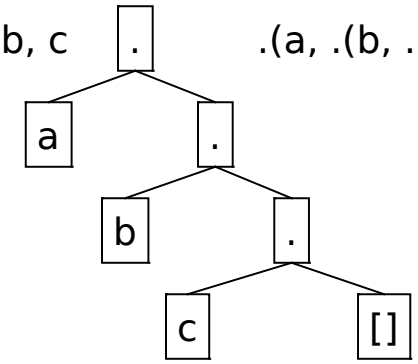
2. Listas

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

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



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



Representación habitual de las listas:

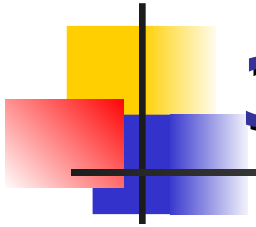
[a,b,c]

[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]



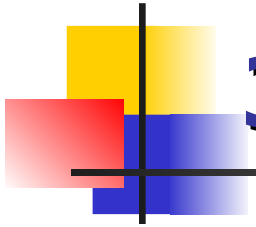
3. Operaciones con Listas

Pertenencia (I)

- Saber si un objeto pertenece a lista.
- Ejemplo:

  [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]

- Se puede construir un predicado “miembro” para saber si una elemento está en una lista o usar el predefinido “**member**”.
  - Probar su doble funcionamiento:
    - Primer argumento instanciado
    - No instanciado



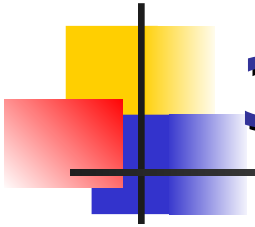
3. Operaciones con Listas

Pertenencia (I)

- Saber si un objeto pertenece a lista.
- Ejemplo:

  [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]

- Se puede construir un predicado “miembro” para saber si una elemento está en una lista o usar el predefinido “**member**”.
  - Probar su doble funcionamiento:
    - Primer argumento instanciado
    - No instanciado



3. Operaciones con listas (II)

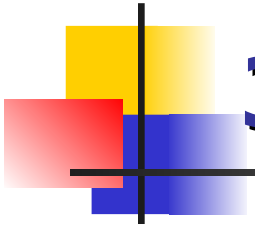
Insertar un elemento

- 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:

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

- Ejemplo:

```
?- insertar2(rojo, [verde, azul], Colores).
   Colores = [rojo, verde, azul].
```



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]).
true.
?- append([a], [1, 2, 3], [alfa, beta, gamma]).
false.
```

- El mismo predicado creado por el usuario:

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

4. Prácticas (II) (hecho FIA)

Definir las siguientes operaciones sobre listas:

1. *longitud(L, Num)*
   Determina el número de elementos de L.
2. *ultimo (Ult, Lista)*
   ultimo elemento de la lista.
3. *Subconjunto(Lista1, Lista2)*
   ¿Es Lista1 un subconjunto de Lista2?

4. Prácticas (III) (FIA)

Ordenación alfabética (I)

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

- Obtener la lista asociada a cada palabra:

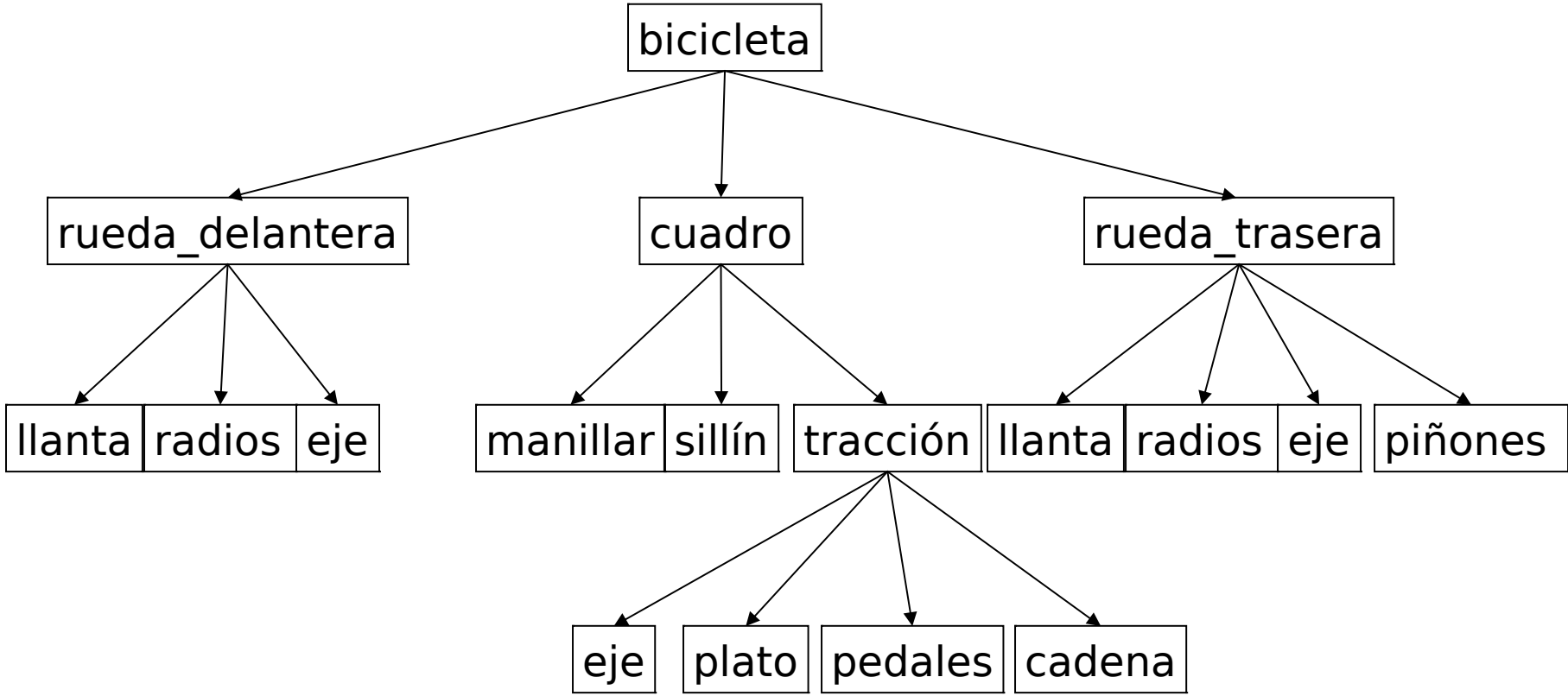
```
?- name(pala, X).
X=[112, 97, 108, 97].
```

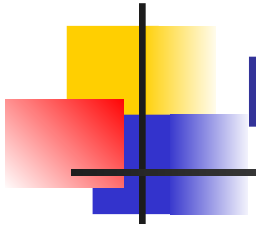
- Comparación:

```
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).
```

4. Prácticas (IV) (ejercicio)

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.