PROLOG - Estructuras de Datos y Listas

Abril 2018

Partes de este material fueron tomadas de:
https://www.infor.uva.es/~calonso/IAI/PracticasProlog/

Contenido

Estructuras y árboles

Listas

Operaciones con Listas

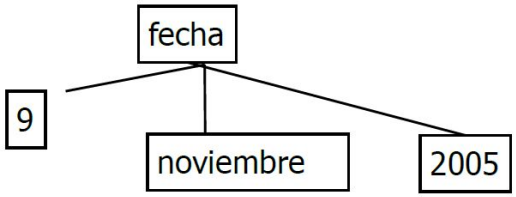
Ejercicios

Prácticas

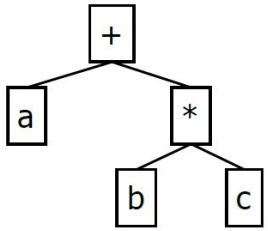
Estructuras y árboles

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

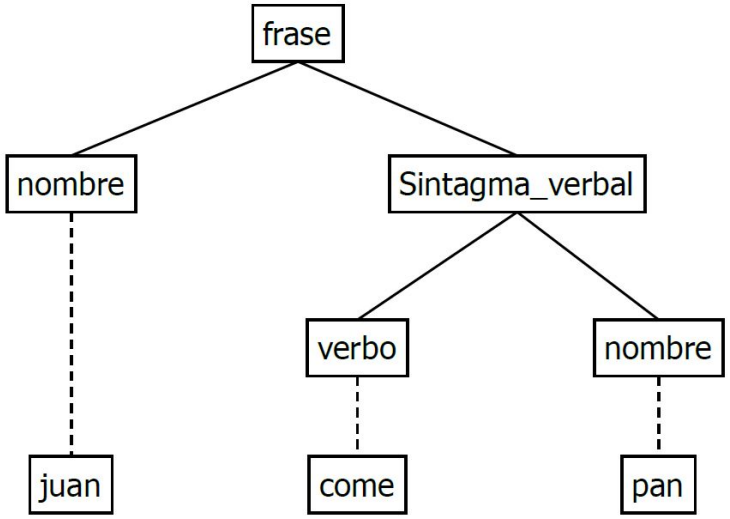
fecha(9, noviembre, 2005)



a+b*c



frase(nombre, sintagma_verbal(verbo, nombre))



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

Contenido

Estructuras y árboles

Listas

Operaciones con Listas

Ejercicios

Prácticas

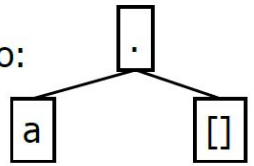
Listas

- 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.

Listas (cont.)

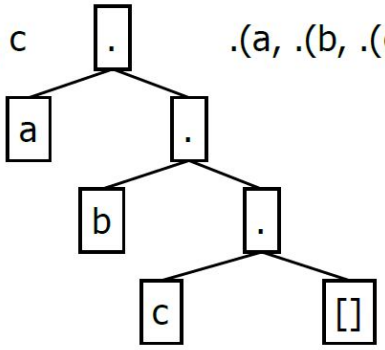
- Tipo particular de árbol: cabeza y cola
  - El functor/estructura asociado es “.”
  - El final de la lista es “[]”

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]

Contenido

Estructuras y árboles

Listas

Operaciones con Listas

Ejercicios

Prácticas

Operaciones con listas (1)

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

Operaciones con listas (1)

Pertenencia (cont)

```
?-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

- Natural en muchas definiciones declarativas
- Perspectiva operacional: natural si la misma operación se aplica a distintos datos

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

Recursión (cont.)

- 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 program

- Evitar definiciones circulares:

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

  - (se entra en un bucle infinito)

Operaciones con listas (2)

Insertar un elemento

- Queremos introducir un elemento X al inicio 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]).`

Operaciones con listas (2)

Insertar un elemento (cont.)

- Ejemplos:

```
?-insertar(1, [3, 5, 7], Primos).
Primos = [1, 3, 5, 7]

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

Operaciones con listas (3)

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).
```

Ejercicio

- Alterar frase
  - Diálogo:
    - Usuario: tu eres un ordenador
    - Prolog: yo no soy un ordenador
    - Usuario: hablas frances
    - Prolog: no_hablo aleman (“_” simula a “,”)
- Reglas:
  - Aceptar frase en formato lista:
  - [tu, eres, un ordenador]
  - Cambiar tu por yo.
  - Cambiar cada eres por un [no_soy].
  - Cambiar cada hablas por un [no_hablo].
  - Cambiar cada frances por un aleman.

Ejercicio - solución

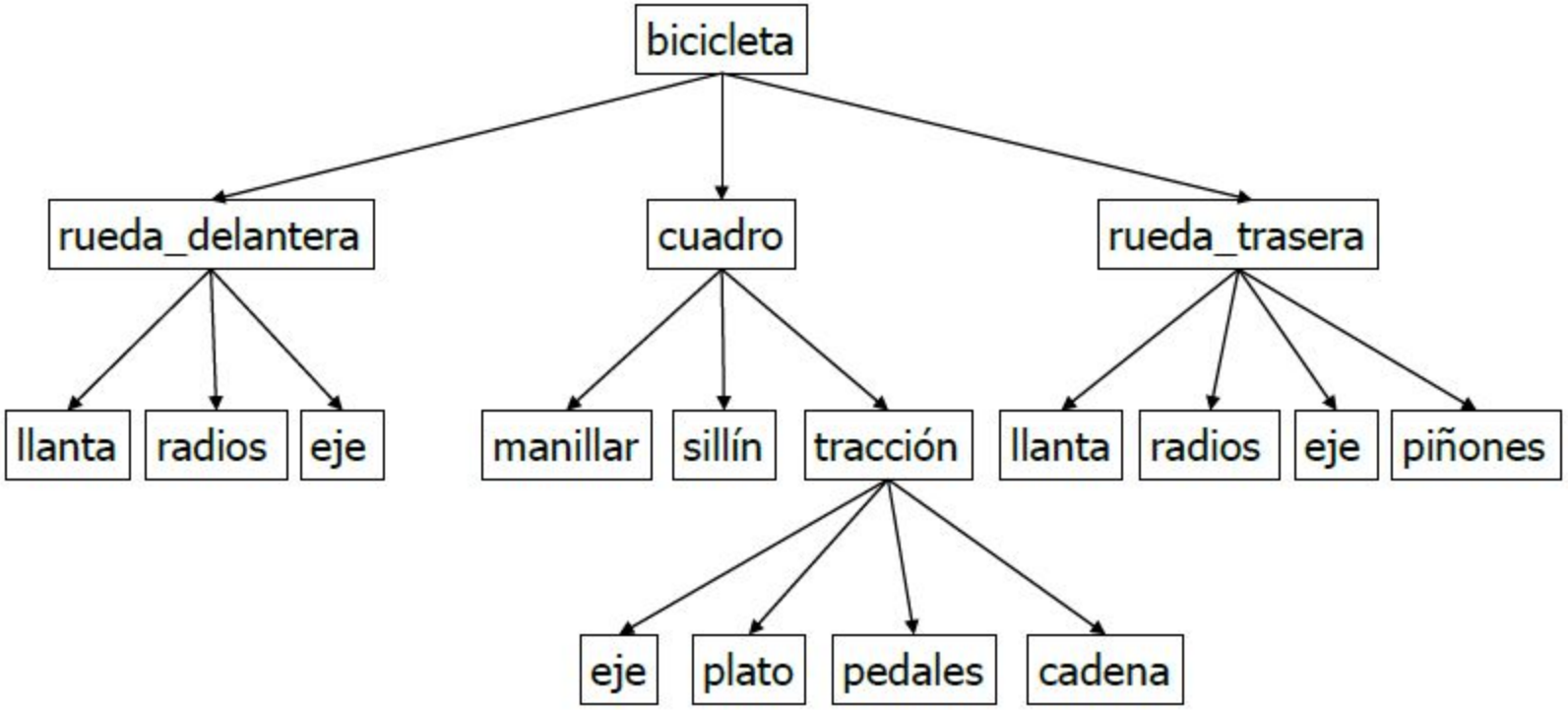
```
cambiar(tu, yo).
cambiar(eres, [no, soy]).
cambiar(hablas, [no_, hablo]).
cambiar(frances, aleman).
cambiar(X,X).
alterar([], []).
alterar([H|T], [X|Y]):-cambiar(H,X), alterar(T,Y)
```

Ejercicio - solución (cont)

```
?-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] ;
```

Práctica

- Lista de Partes



Práctica (cont.)

- Definir el árbol mediante las relaciones:
  - `parte(cadena).`
  - `ensamble(bicicleta, [rueda_delantera, cuadro, rueda_trasera]).`
- Construir relaciones “partes”, que sirva para obtener la lista de artes para construir un ensamble (o toda) la bicicleta.

Práctica (cont.)

```
parte(llanta).
parte(radios).
parte(eje).
parte(manillar).
parte(sillin).
parte(pinon).
parte(plato).
parte(pedales).
parte(cadena).

ensamble(rueda_delantera, partes([llanta, radios, eje])).
ensamble(cuadro, partes([manillar, sillin, traccion])).
ensamble(rueda_trasera, partes([llanta, radios, eje, pinon])).
ensamble(traccion, partes([eje, plato, pedales, cadena])).
ensamble(bicicleta, partes([rueda_delantera, cuadro, rueda_trasera])).
```

Práctica (cont.)

```
concatenar([], L, L).
concatenar([X|Y], Z, [X|U]) :- concatenar(Y, Z, U).

partesde(X, [X]) :- parte(X).
partesde(X, P) :- ensamble(X, partes(Partes)), listapartes(Partes, P).

listapartes([], []).
listapartes([H|T], Lista) :- partesde(H, PartesH),listapartes(T,
PartesT),concatenar(PartesH, PartesT, Lista).
```

Práctica (cont.)

```
concatenar([], L, L).
concatenar([X|Y], Z, [X|U]) :- concatenar(Y, Z, U).

partesde(X, [X]) :- parte(X).
partesde(X, P) :- ensamble(X, partes(Partes)), listapartes(Partes, P).

listapartes([], []).
listapartes([H|T], Lista) :- partesde(H, PartesH),listapartes(T,
PartesT),concatenar(PartesH, PartesT, Lista).
```

Práctica (cont.)

```
?- partesde(cuadro,K).

K = [manillar, manillar, sillin, sillin, traccion, eje, eje, plato,
plato|...]
```