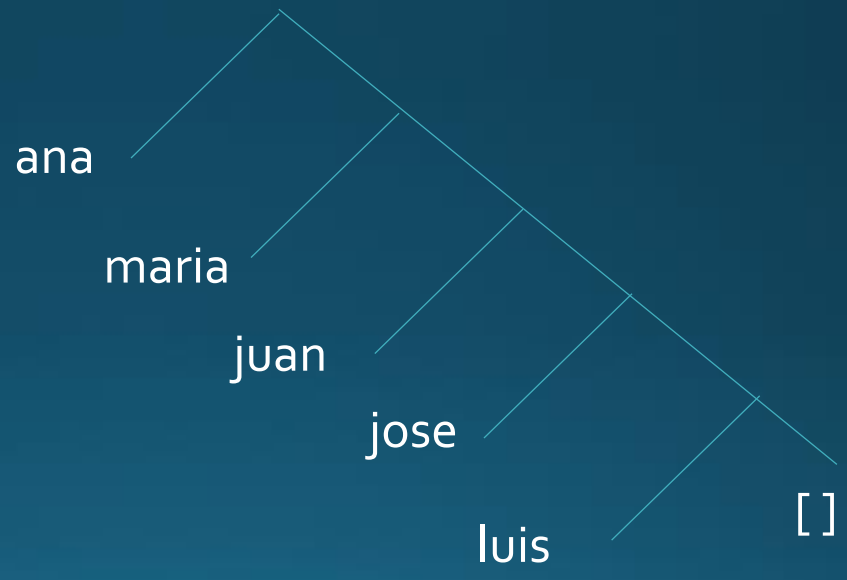
Listas

Listas

- Una lista es una secuencia ordenada de elementos que puede tener cualquier longitud.
- Los elementos de una lista pueden ser cualquier término (constantes, variables, estructuras) u otras listas.

Listas

- [ana, maria, juan, jose, Luis]
- En Prolog son arboles binarios



Listas

- Una lista es una secuencia finita de elementos
- Una lista puede definirse recursivamente como:
  - Una lista vacía [], sin elementos.
  - Una estructura con dos componentes:
    - cabeza: primer elemento.
    - cola: el resto de la lista.
- Ejemplos:
  - L = [jose, pedro, maria]
  - L = [camino(rg,co), [b, c, d], Z]
  - L = [pepe, []]

Lista

- La lista vacía no tiene ni cabeza ni cola.
- Para Prolog, [ ] es una lista simple especial sin ninguna estructura interna.
- La lista vacía juega un papel importante en los predicados recursivos para el procesamiento de listas en Prolog.
- Términos
  - Lista vacia: [].
  - Lista compuesta: [<termino>+f| <lista>g].
    - <términos>: sucesión de los primeros elementos de la lista.
    - <lista>: lista con los restantes elementos.
    - [a, b, c] = [a, b, c| []] = [a, b| [c]] = [a|[b, c]].

Listas

- La operación fundamental es la separación de la cabeza de la cola mediante unificación.
  - [X|Y] = [a, b, c, d, e] → X=a , Y=[b, c, d, e]
  - [X|Y] = [[a, b], c, d] → X=[a, b] , Y=[c, d]
  - [X|Y] = [a] → X=a , Y=[]
  - [X|Y] = [a, b] → X=a , Y=[b]

Operadores

- Prolog tiene un operador especial integrado | que se puede utilizar para descomponer una lista en su cabeza y cola.
- El | operator es una herramienta clave para escribir predicados de manipulación de listas Prolog
- Ej. ?- [X|Y] = [mia, vincent, jules, yolanda].
  - X = mia
  - Y = [vincent,jules,yolanda]
- ?- [X|Y] = [ ].
  - no

Variable anónima

- Supongamos que estamos interesados en el segundo y cuarto elemento de una lista.
- ?- [X1,X2,X3,X4|Cola] = [pedro, luis, soledad, marcelo, yolanda].
  - X1 = pedro
  - X2 = luis
  - X3 = soledad
  - X4 = marcelo
  - Tail = [yolanda]

Variable anónima

- Hay una forma más sencilla de obtener solo la información que queremos:
- ?- [_,X2, _,X4|_] = [pedro, luis, soledad, marcelo, yolanda].
- X2 = luis
- X4 = yolanda
- El guión bajo es la variable anónima

Variable anónima

- Se utiliza cuando necesita usar una variable, pero no está interesado en lo que Prolog instancia.
- Cada ocurrencia de la variable anónima es independiente, es decir, puede estar ligada a algo diferente.

Ejercitemos

- ¿Cómo responde Prolog a las siguientes consultas?
  - [a,b,c,d] = [a,[b,c,d]].
  - [a,b,c,d] = [a|[b,c,d]].
  - [a,b,c,d] = [a,b,[c,d]].
  - [a,b,c,d] = [a,b|[c,d]].
  - [a,b,c,d] = [a,b,c,[d]].
  - [a,b,c,d] = [a,b,c|[d]].
  - [a,b,c,d] = [a,b,c,d,[]].
  - [a,b,c,d] = [a,b,c,d|[]].
  - [] = _.
  - [] = [_].
  - [] = [_|[]]

Listas

- Si se tiene la lista [a, b, c, d], la a es la cabeza y la cola es la lista [b, c, d]
  - Una lista cuya cabeza es A y cola es B se anota como *A | B+
  - El predicado:
    - primer_elemento(X, [X|_]).
    - tiene éxito si X es el primer elemento de la lista.
- Ejemplo:
  - ?- [1,2,3] = [1|[2|[3|[]]]].
  - ?- [X|Xs]=[1,2,3].

    X = 1,

    Xs = [2, 3] .

Ejercitemos

- ¿¿Cuáles de las siguientes son listas sintácticamente correctas?
- Si la representación es correcta, ¿cuántos elementos tiene la lista?
  1. [1|[2,3,4]]
  2. [1,2,3|[]]
  3. [1|2,3,4]
  4. [1|[2|[3|[4]]]]
  5. [1,2,3,4|[]]
  6. [[]|[]]
  7. [[1,2]|4]
  8. [[1,2],[3,4]|[5,6,7]]

Member

- Una de las cosas más básicas que nos gustaría saber es si algo es un elemento de una lista o no.
- Así que escribamos un predicado que cuando se le da un término X y una lista L, nos dice si X pertenece o no a L
- Este predicado generalmente se llama **member.**
  - member(X,[X|T]).
  - member(X,[H|T]):- member(X,T).
    - ?- member (pedro,[pedro,luis, soledad, marcelo, yolanda]).
    - ?- member (pedro,[luis, soledad, pedro, marcelo, yolanda]).
    - ?- member (pedro,[luis, soledad,marcelo,yolanda]).
    - ?- member (X,[luis,soledad,marcelo,yolanda]).
  - member(X,[X|_]).
  - member(X,[_|T]):- member(X,T).

Listas recursivas

- El predicado member funciona trabajando recursivamente en una lista
  - haciendo algo a la cabeza,
  - y luego recursivamente haciendo lo mismo con la cola

Hagamos el predicado **a2b**

- El predicado a2b/2 deberá tomar dos listas como argumentos y tiene éxito
  - si el primer argumento es una lista de a, y
  - El segundo argumento es una lista de B de
- exactamente la misma longitud.
- ?- a2b([a,a,a,a],[b,b,b,b]).
  - true
- ?- a2b([a,a,a,a],[b,b,b]).
  - false
- ?- a2b([a,c,a,a],[b,b,b,t]).
  - true

Hagamos el predicado a2b

- a2b([],[]).
  - A menudo, la mejor manera de resolver tales problemas es pensar en el caso más simple posible. Aquí significa: la lista vacía.
- a2b([a|L1],[b|L2]):- a2b(L1,L2).
  - ¡Ahora piensa recursivamente! ¿Cuándo debería a2b decidir que dos listas no vacías son una lista de a y una lista de b de exactamente la misma longitud?
- ?- a2b([a,a,a,a],[b,b,b,b]).
- ?- a2b([a,a,a,a],[b,b,b]).
- ?- a2b([a,c,a,a],[b,b,b,t]).
- ?- a2b([a,a,a,a,a], X).
- ?- a2b(X,[b,b,b,b,b,b,b]).

Append

- Define append, un predicado para concatenar dos listas, e ilustrar lo que se puede hacer con ella
- Veremos dos formas de invertir una lista
  - Una forma simple usando append.
  - Un método más eficiente utilizando acumuladores

Append

- Definiremos un predicado importante **Append** cuyos argumentos son todas listas.
  - Declarativamente, append(L1,L2,L3) es true
  - si la lista L3 es el resultado de concatenar las listas L1 y L2 juntas.
- ?- append([a,b,c,d],[3,4,5],[a,b,c,d,3,4,5]).
- ?- append([a,b,c],[3,4,5],[a,b,c,d,3,4,5]).

Append

- Desde una perspectiva procesal, el uso más obvio de append es concatenar dos listas juntas.
- Podemos hacer esto simplemente usando una variable como tercer argumento
- ?- append([a,b,c,d],[1,2,3,4,5], X).

Append

- Definición recursiva
  - Cláusula base: añadir la lista vacía a cualquier lista produce esa misma lista
  - El paso recursivo dice que al concatenar una lista no vacía [H|T] con una lista L, el resultado es un lista con cabeza H y el resultado de concatenar T y L.

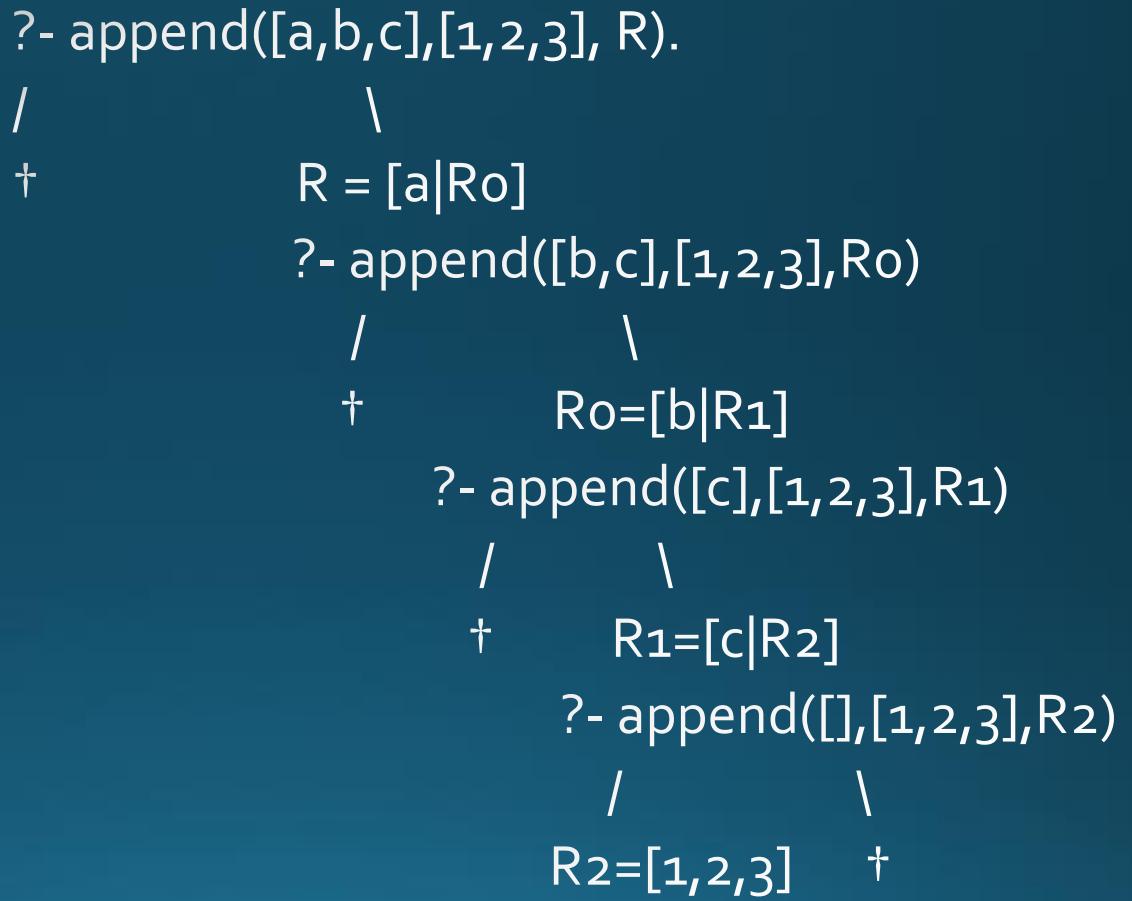
```
append([ ], L, L).
append([H|L1], L2, [H|L3]):- append(L1, L2, L3).
```

Append

- Dos formas de averiguarlo:
  - Usar trace/o en algunos ejemplos
  - ¡Dibuja un árbol de búsqueda!
  - Consideremos un ejemplo simple

- append([], L, L).
- append([H|L1], L2, [H|L3]):-
- append(L1, L2, L3).

- R2=[1,2,3]
- R1=[c|R2]=[c,1,2,3]
- R0=[b|R1]=[b,c,1,2,3]
- R=[a|R0]=[a,b,c,1,2,3]



Append

- Ahora que entendemos cómo funciona el append, veamos algunas aplicaciones.
- Dividir una lista:
  - ?- append(X,Y, [a,b,c,d]).
  - X=[ ]Y=[a,b,c,d];
  - X=[a]Y=[b,c,d];
  - X=[a,b]Y=[c,d];
  - X=[a,b,c]Y=[d];
  - X=[a,b,c,d]Y=[ ];
  - False

Prefijo y Sufijo

- También podemos usar append para definir otros predicados útiles. Un buen ejemplo es encontrar prefijos y sufijos de una lista.
- **Definición de prefix**
  - Una lista P es un prefijo de alguna lista L cuando hay alguna lista tal que L es el resultado de concatenar P con esa lista.
  - Usamos la variable anónima porque no nos importa cuál es esa lista.
    - prefix(P,L):- append(P,_,L).
  - ?- prefix(X, [a,b,c,d]).
    - X=[ ];
    - X=[a];
    - X=[a,b];
    - X=[a,b,c];
    - X=[a,b,c,d];
    - false

Prefijo y Sufijo

- **Definición de sufix**
- Una lista S es un sufijo de alguna lista L cuando hay alguna lista tal que L es el resultado de concatenar esa lista con S.
- Una vez más, usamos la variable anónima porque no nos importa cuál es esa lista.
  - suffix(S,L):- append(_,S,L).
  - ?- suffix(X, [a,b,c,d]).
    - X=[a,b,c,d];
    - X=[b,c,d];
    - X=[c,d];
    - X=[d];
    - X=[];
    - no

Sublista

- Ahora es muy fácil escribir un predicado que encuentre sublistas de listas.
- Las sublistas de una lista L son simplemente los prefijos de los sufijos de L
    - sublist(Sub,List):- suffix(Suffix,List), prefix(Sub,Suffix).

append y eficiencia

- El predicado append es útil, y es importante saber cómo usarlo
- Es de igual importancia saber que append puede ser fuente de ineficiencia
- ¿Por qué?
  - Concatenar una lista no se realiza en una simple acción
  - Pero recorriendo una de las listas
- Usando append nos gustaría concatenar dos listas:
  - List 1: [a,b,c,d,e,f,g,h,i]
  - List 2: [j,k,l]
- El resultado debe ser una lista con todos los elementos de la lista 1 y 2, el orden de los elementos no es importante
- ¿Cuál de los siguientes objetivos es la forma más eficiente de concatenar las listas?
  - ?- append([a,b,c,d,e,f,g,h,i],[j,k,l],R).
  - ?- append([j,k,l],[a,b,c,d,e,f,g,h,i],R).

append y eficiencia

- Observe la forma en que se define append.
- Se repite en el primer argumento, sin tocar realmente el segundo argumento.
- Eso significa que es mejor llamarlo con la lista más corta como primer argumento.
- Por supuesto, no siempre sabes cuál es la lista más corta, y solo puedes hacerlo cuando no te importa el orden de los elementos en la lista concatenada.
- Pero si lo hace, puede ayudar a que su código Prolog sea más eficiente.

append y eficiencia

- Observe la forma en que se define append.
- Se repite en el primer argumento, sin tocar realmente el segundo argumento.
- Eso significa que es mejor llamarlo con la lista más corta como primer argumento.
- Por supuesto, no siempre sabes cuál es la lista más corta, y solo puedes hacerlo cuando no te importa el orden de los elementos en la lista concatenada.
- Pero si lo hace, puede ayudar a que su código Prolog sea más eficiente.

Invertir una lista

- Mostramos el problema con append usándolo para invertir elementos de una lista.
- Es decir, definiremos un predicado que cambia una lista [a,b,c,d,e] en una lista [e,d,c,b,a].

Inversa de Naïve

- Definición recursiva
  - Si invertimos la lista vacía, obtenemos la lista vacía
  - Si invertimos la lista [H|T], terminamos con el lista obtenida invirtiendo T y concatenando con [H].
- Para ver que esta definición es correcta, considere la lista [A,B,C,D].
  - Si invertimos la cola de esta lista obtenemos [d,c,b].
  - Concatenando esto con [a] se obtiene [d,c,b,a]

- accReverse([ ],L,L).
- accReverse([H|T],Acc,Rev):- accReverse(T,[H|Acc],Rev).

- reverse(L1,L2):- accReverse(L1,[ ],L2).

Ejercitemos

- Llamemos a una lista duplicada si está hecha de dos bloques consecutivos de elementos que son exactamente iguales. Por ejemplo, [a,b,c,a,b,c] se duplica (se compone de [a,b,c] seguido de [a,b,c]) y también lo es [hola,chau,hola,chau] . Por otro lado, [hola, chau, hola] no se duplica. Escriba un predicado doblel(Lista) que se realiza correctamente cuando Lista es una lista duplicada.
- Un palíndromo es una palabra o frase que deletrea lo mismo hacia adelante y hacia atrás. Por ejemplo, 'ana', 'somos' y yo hago yoga hoy ' son todos palíndromos. Escriba un predicado palindrome(Lista) , que comprueba si Lista es un palíndromo. Por ejemplo, a las consultas
  - ?- palindrome([a,n,a])
  - ?- palindrome([s,o,m,o,s]).
  - Prolog debería responder True, pero a la consulta
  - ?- palindrome([h,o,y]).
  - Debera responder False

Ejercitemos

- Escriba un predicado toptail(InList,OutList) que diga no si InList es una lista que contiene menos de 2 elementos, y que elimina el primer y el último elemento de InList y devuelve el resultado como OutList , cuando InList es una lista que contiene al menos 2 elementos. Por ejemplo
  - toptail([a],T).
    - False
  - toptail([a,b],T).
    - T=[]
  - toptail([a,b,c],T).
    - T=[b]
  - (Sugerencia: aquí es donde append es útil).
- Escriba un predicado last(List,X) que es true sólo cuando List es una lista que contiene al menos un elemento y X es el último elemento de esa lista. Haga esto de dos maneras diferentes:
  - Defina last usando el predicado rev discutido en el texto.
  - Defina last usando recursión.
  - ?- last([a, b, c], X).
    - X = c.
  - ?- last([x], X).
    - X = x.

Ejercitemos

- Escriba un predicado swapfl(List1,List2) que compruebe si List1 es idéntico a List2 , excepto que se intercambian el primer y el último elemento. Aquí es donde append podría ser útil de nuevo, pero también es posible escribir una definición recursiva sin apelar a append (o cualquier otro) predicado.
  - ?- swapfl([a, b, c, d, e], [e, b, c, d, a]).
    - true.
  - ?- swapfl([a, b, c], [c, b, a]).
    - true.
  - ?- swapfl([a, b, c], [a, b, c]).
    - false.