



Tecnologías de Programación

Paradigma Lógico – ProLog

Listas



Lista en Prolog

- La lista es una estructura de datos simple y representa una secuencia de elementos.
 - Ej.: juan, manuel, maría.
- En PROLOG las listas se representan como la secuencia de elementos que contiene, separada por “,” (coma) y encerrada entre corchetes.
 - Ej.: [juan, manuel, maría]
- La lista vacía se representa de la siguiente forma: “[]”.

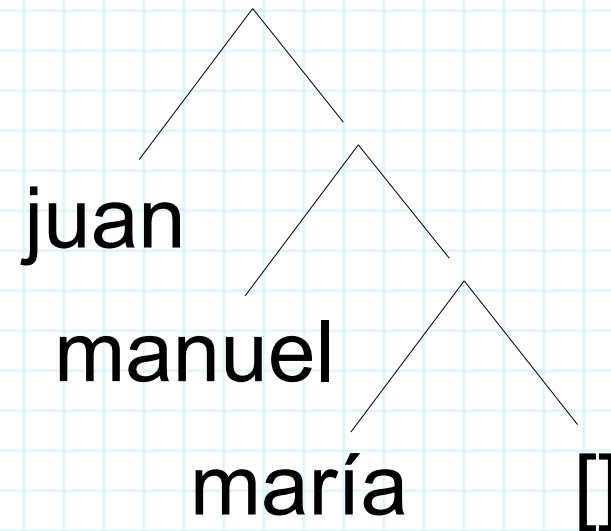


Representación Interna

- Internamente las listas en Prolog se representan como un árbol binario, donde la rama de la izquierda es el primer elemento –o cabeza– de la lista y la rama de la derecha es el resto –o cola– de la lista.
- El último elemento de la lista, se representa con el elemento como cabeza y una lista vacía como cola.

Representación Interna

- La representación en forma de árbol de la lista sería:



- Siempre se considerará dos casos de listas: la vacía (el átomo “[]”) y las no vacías, formadas por dos partes (la cabeza y la cola).



Functor Lista (.)

- Aunque es poco usado, ya que la representación vista anteriormente es más práctica, la estructura “lista” se puede representar en PROLOG a través del functor “.”
.
- Así la lista [juan, manuel, maría] se puede representar también como:
.(juan, .(manuel, .(maría, [])))



Elementos de la Lista

- Una lista puede contener elementos de cualquier tipo y éstos elementos no necesitan ser de tipos homogéneos, es decir cada elemento en particular puede ser de cualquier tipo (incluyendo otras listas).
 - `[1, a, José, 1.23, [1, 2], 2]`
- El orden de los elementos es importante:
 - `[1, 2, 3] != [1, 3, 2]`
- La lista puede contener elementos repetidos:
 - `[1, 1, 2, 2, 3, 3]`

Ejercicios 1 (25 min.)

- Represente en forma de árbol binario las siguientes listas:

1) [1, 2, 3]

2) []

3) [[]]

4) [a, [b, c], [d, e]]

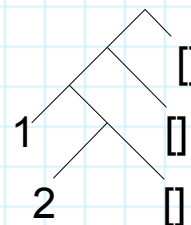
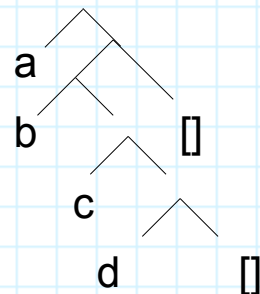
- Represente las listas usando el functor “.”:

1) ['Buenos Aires', 'Santa Fe', 'Entre Ríos', 'Misiones']

2) [[1, 2], [1, 3], [6, 3]]

3) [3.23, []]

- Representar los siguientes árboles binarios como listas:





Representación de Listas

- Como hemos visto, internamente una lista no vacía, se puede representar como un átomo que conforma la cabeza y una lista que conforma el cuerpo.
- En Prolog se puede utilizar el pipe “|” como símbolo para separar los primeros N elementos, del resto de la lista:
 - Ej.: [1, 2, 3] es equivalente a [1 | [2, 3]]



Representación de Listas

- Estas representaciones corresponden a la misma lista:
 - $[a, b, c]$
 - $[a \mid [b, c]]$
 - $[a, b \mid [c]]$
 - $[a, b, c \mid []]$



Manejo de Listas

- En PROLOG el manejo de listas constituye uno de los conceptos básicos y más utilizados.
- Para trabajar con listas, generalmente se utiliza la recursividad, mediante el pipe “|” se pueden separar los primeros elementos a utilizar, de lo que quede en la lista a procesar en próximas invocaciones.



Manejo de Listas

- Supongamos que queremos realizar un predicado por el cual se determine si un átomo es miembro de una lista.
- Una forma de pensarlo podría ser:
 - Un átomo pertenece a una lista si:
 - unifica con el primer elemento de la lista (cabeza).
 - o si es miembro de la cola no vacía de la lista.
 - En prolog:
 - pertenece(X, [X | _]).
 - pertenece(X, [_ | C]) :- pertenece(X, C).



Ejercicios 2 (30 min.)

1) Ingrese al editor de Prolog (JprologEditor), transcriba el ejemplo anterior y ejecute las siguiente preguntas:

- a) pertenece(3, [1, 2, 3]).
- b) pertenece(a, [1, 2, 3]).
- c) pertenece(X, [1, 2, 3]).
- d) pertenece(2, [1, X, 3]).

Una vez probados y obtenidos los resultados, discuta las respuestas para obtener conclusiones de como se ha arribado a los mismos

2) Dibuje el árbol de resolución para cada una de las preguntas.



Manejo de Listas

- Suponemos que queremos concatenar dos listas y devolver el resultado en una tercera:
 - `concatenar([a, b, c], [d, e], [a, b, c, d, e]).` -- `true`
 - `concatenar([a, b], [1, 2, 3], X).` -- `X = [a, b, 1, 2, 3].`
- Una forma de plantearlo sería:
 - La concatenación de las listas es:
 - La segunda lista, si la primera es la lista vacía.
 - Una lista conformada por:
 - Primer elemento: primer elemento de la primer lista.
 - Cola: la cola de la primer lista concatenada con la segunda lista.
- La implementación sería:
 - `concatenar ([], L, L).`
 - `concatenar ([X | L1], L2, [X | L3]) :- concatenar (L1, L2, L3).`



Ejercicios 2 (30 min.)

1) Ingrese al editor de Prolog (JprologEditor), transcriba el ejemplo anterior y ejecute las siguiente preguntas:

- a) concatenar([a, b], [1, 2], Lista).
- b) concatenar([a, b], [1, 2], [a, b]).
- c) concatenar(X, [1, 2], [1, 2]).
- d) concatenar(X, [1, 2], [a, b, 1, 2]).

Una vez probados y obtenidos los resultados, discuta las respuestas para obtener conclusiones de como se ha arribado a los mismos.

2) Dibuje el árbol de resolución para cada una de las preguntas.



Manejo de Listas

- Suponemos que queremos invertir una lista:
 - `invertir([a, b, c], [c, b, a]).`
 - `true`
 - `invertir([a, b], Lista).`
 - `Lista = [b, a].`
- La implementación sería:
 - `invertir([], []).`
 - `invertir([X | L1], L):-`

`invertir(L1, Resto),`
`concatenar(Resto, [X], L).`



Expresiones Aritméticas

- Los siguientes operadores y predicados predefinidos pueden usarse para construir expresiones aritméticas:

Suma: $X + Y$

Resta: $X - Y$

Producto: $X * Y$

División: X / Y

Módulo: $X \bmod Y$

Potencia: $X ^ Y$

Negación: $-X$

Valor absoluto: $\text{abs}(X)$

Arco coseno: $\text{acos}(X)$

Arco seno: $\text{asen}(X)$

Arco tangente: $\text{atan}(X)$

Coseno: $\text{cos}(X)$

Exponencial: $\text{exp}(X)$

Log. neperiano: $\text{ln}(X)$

Log. base 2: $\text{log}(2)$

Seno: $\text{sin}(X)$

Raíz cuadrada: $\text{sqrt}(X)$

Tangente: $\text{tan}(X)$



Comparación de Términos

- Los siguientes operadores permiten comparar y unificar términos:
 - $X = Y$ % verdadero si X unifica con Y
 - $X \neq Y$ % verdadero si X no unifica con Y



Orden de Términos

- Comparación y unificación de términos arbitrarios. Los términos se ordenan según el siguiente criterio:
 - 1. Variables < Números < Átomos < Strings < Estructuras
 - 2. Las variables son ordenadas por dirección
 - 3. Los átomos son comparados alfabéticamente
 - 4. Las cadenas son comparadas alfabéticamente
 - 5. Los números se comparan por valor, en caso de igualdad, el real se considera menor
 - 6. Las estructuras son validadas primero por su aridad, luego por su nombre y finalmente en forma recursiva por sus argumentos de izquierda a derecha
- Es igual a: $X == Y$
- Es distinta de: $X \neq Y$
- Es menor / menor o igual a: $X @< Y$ / $X @ \leq Y$
- Es mayor / mayor o igual a: $X @> Y$ / $X @ \geq Y$



Evaluación de Expresiones

- En Prolog NO EXISTE al asignación de valores, una variable queda ligada solo por unificación.
- De acuerdo a lo siguiente, supongamos que tenemos la variable libre "X". La expresión "X = 1" ligaría el valor de "X" a "1", pero no porque se le este asignando, sino porque es el único valor que puede tomar "X" para hacer verdadera la expresión.
- Supongamos que queremos unificar "X" con el resultado de una expresión, "X = 1 + 2" unificaría "X" con la expresión "1 + 2" y no con el resultado de la misma "3"
- El operador "is" permite unificar lo que hay del lado izquierdo con el resultado de evaluar aritméticamente lo que haya en el lado derecho. Así "X is 1 + 2" unificará "X" con el término "3"
- De lo dicho anteriormente se desprende que en Prolog la siguiente expresión no tiene sentido:
 - X is X + 1 % ¿por que?



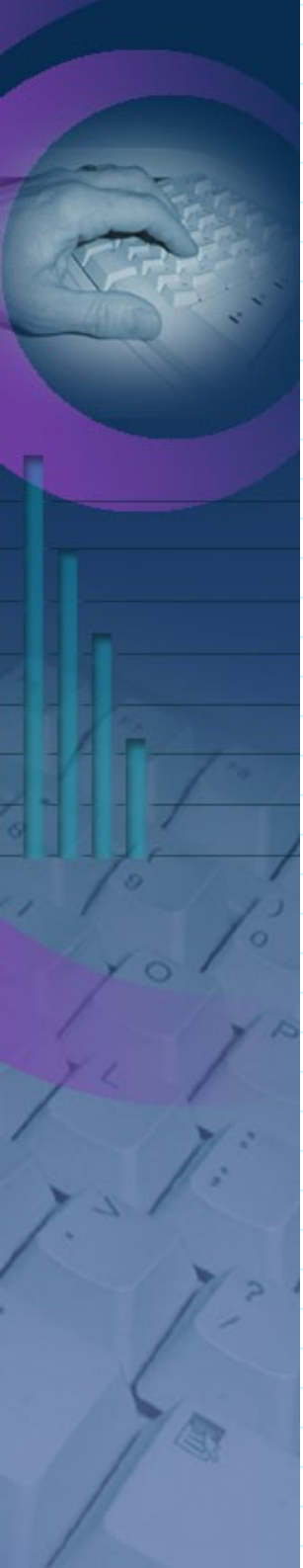
Evaluación de Expresiones

- Los siguiente operadores comparan términos haciendo una evaluación aritmética de las expresiones que los componen:
 - El resultado de evaluar aritméticamente las expresiones es el mismo: $X ::= Y$
 - El resultado de evaluar aritméticamente las expresiones es distinto: $X \neq Y$



Ejemplos

- Predicado en Prolog para determinar la longitud de una lista:
 - Ej.: `longitud([a, b, c], X).` -- $X = 3$
 - Ej.: `longitud([a, b, c], 3).` -- True
- El planteo sería:
 - La longitud de la lista vacía es 0 (cero).
 - La longitud de una lista no vacía es $1 +$ la longitud de la cola de dicha lista.
- La implementación sería:
 - `longitud([], 0).`
 - `longitud(_ | C, L) :- longitud(C, L1), L is L1 + 1.`



Ejercicios 3 (30 min.)

- 1) Escribir el predicado `cantpos/2` para determinar la cantidad de elementos positivos de una lista de números (considerar el cero como positivo).

Ej: `cantpos([1, 2, 0, -1, -5, 2], X).`

`X = 4`

- 2) Escribir un predicado `separar/3` que permita a partir de una lista de números, obtener dos listas, una con los números positivos y otra con los negativos.

Ej.: `separar([1, 2, 0, -1, -5, 2], Pos, Neg).`

`Pos = [1, 2, 0, 2]`

`Neg = [-1, -5]`

- 3) Dada una lista de números, escribir el predicado `maxmin/3` que permita obtener el máximo y el mínimo de la lista.

Ej.: `maxmin([1, 2, 0, -1, -5, 2], Max, Min).`

`Max = 2`

`Min = -5`