5.1 Listas

En este capítulo introducimos el tipo de dato más importante dentro de la programación en inteligencia artificial. Existe un lenguaje de programación llamado LISP (LISt Procesing), en el que las únicas estructuras de datos disponibles son las constantes y las listas. En Prolog, la **lista** es simplemente un tipo concreto de estructura.

¿Por qué usar listas?

Las listas son una manera fácil de trabajar en situaciones imprevisibles. Se utilizan en situaciones donde es difícil o imposible predecir el número de datos a ser guardados o manipulados. En muchos problemas de inteligencia artificial, el número de soluciones parciales a un problema cambia de forma dinámica durante la ejecución.

¿Qué es una lista?

En Prolog, una lista es un objeto que contiene un número arbitrario de otros objetos, cada uno de estos objetos se llaman **elementos** de la lista. He aquí una lista de enteros

$$[1, 2, 3, 4, 5]$$

Los elementos de una lista pueden ser de cualquier tipo de dato, incluso pueden ser a su vez listas

[[juan,pepe,luis],[juana,pepa,luisa]]

La lista que no contiene ningún elementos es la **lista vacía** [ ].

¿Cómo se declaran ?

La declaración en su forma general es

domains

listacosas = cosas*

cosas = objeto

Puede ser más sencilla esta declaración si es una lista de un tipo de dato predefinido.

domains

listaenteros = integer*

¿Cómo se manipulan?

Las listas se manipulan dividiéndolas en una **cabeza** y una **cola**. Una lista es realmente un objeto compuesto recursivo, que consiste en la cabeza que es el primer elemento, y la cola, que es a su vez una lista que contiene todos los elementos menos el primero.

Ejercicio 5.1

Lista	Cabeza	Cola
$[a, b, c]$	a	$[b, c]$
$[a]$		
$[\,]$		
[[el, gato], maullo]		
[el, [gato, maullo]]		

5.2 Unificación y Listas

Se puede unificar una lista con otra:

$$[X, Y, Z] = [a, b, c]$$
$$X = a$$
$$Y = b$$
$$Z = c$$

Como una variable que no está instanciada se puede unificar con cualquier objeto, podemos unificar una lista con una variable.

$$X = [a, b, c]$$

Debido a que una operación común con las listas es separar una lista en su cabeza y su cola, existe una notación especial en Prolog para representar **la lista con cabeza X y cola Y**. Esto se escribe $[X\|Y]$, donde el símbolo que separa X de Y es la barra vertical. Una expresión de esta forma, instanciará X a la cabeza de una lista e Y a la cola de la lista:

$$[a, b, c] = [X|Y]$$
$$X = a$$
$$Y = [b, c]$$

$$[a, b, c] = [X, Y|Z]$$
$$X = a$$
$$Y = b$$
$$Z = [c]$$

$$[a, b, c] = [X, Y, Z|Cola]$$
$$X = a$$
$$Y = b$$
$$Z = c$$
$$Cola = [\,]$$

Ejercicio 5.2 Unificar las siguientes listas indicando qué variables quedan instanciadas a qué valores:

[Cabeza | Cola] = [juan, come, pan]

[Cabeza | Cola] = [julia, es, una, enfermera]

[alguien escribe | Cola] = [Cabeza | estas, notas]

[sevilla, unificar] = [sevilla, [Ciudad]]

[sevilla, [unificar,vincular,buscar]] = [Cabeza | Cola]

[sevilla, [unificar,vincular,buscar]] = [Cabeza , Cola]

[nuevo, [ ]] = [Coche, Blanco, Cuento]

[nuevo] = [Cabeza,Cola]

[nuevo] = [Cabeza | Cola]

Debido a que una lista es una estructura recursiva, para utilizarla se necesitan algoritmos recursivos.

Ejercicio 5.4 Haz un predicado **escribir(lista)** que escriba los elementos de una lista.

Ejercicio 5.5 Haz un predicado **pertenece(integer,lista)** que verifique si integer se un elemento de la lista.

5.3 Listas y Recursión

Existen tres criterios de terminación muy importantes

1. Cuando la lista es vacía.
2. Cuando un elemento es encontrado.
3. Cuando una posición es encontrada.

5.3.1 Cuando la lista es vacía

El esquema general es:

/* Regla de terminación */

```
predicado([ ]):- procesar([ ]).
```

/* Regla recursiva */

```
predicado([Cabeza | Cola]):- procesar(Cabeza), predicado(Cola).
```

Ejercicio 5.6 Escribe un predicado **cuenta_elementos(L,N)** de tal forma que N sea el número de elementos que tiene la lista L.

Ejercicio 5.7 Escribe un predicado **suma_lista(L,N)** de tal forma que si L es una lista de enteros, N es la suma de todos ellos.

Ejercicio 5.8 Escribe un predicado **copia(L1,L2)**, de tal forma que L2 es una copia de la lista L1.

Ejercicio 5.9 Escribe un predicado **concatenar(L1,L2,L3)**, de tal forma que L3 es la concatenación de las listas L1 y L2.

5.3.2 Cuando un elemento es encontrado

El esquema general es:

/* Regla de terminación */

```
predicado(Cabeza, [Cabeza | Cola]):- procesar_algo.
```

/* Regla recursiva */

```
predicado(X, [Cabeza | Cola]):- procesar_algo, predicado(X, Cola).
```

Ejercicio 5.10 Escribe un predicado **reemplaza(E1,L1,E2,L2)**, que es verdad si E1 en la lista L1 es reemplazado por E2 para dar L2.

```
reemplaza(1,[1,2,3],4,Lista)
Lista = [4,2,3]

reemplaza(3,[1,2,3],5,Lista)
Lista = [1,2,5]

reemplaza(1,Lista,3,[2,4,3])
Lista = [2,4,1]

reemplaza(1,[ ],3,Lista)
No Solution
```

Ejercicio 5.11 Escribe un predicado **i_despues(E1,L1,E2,L2)**, que es verdad si la lista L2 es la lista L1 con el elemento E1 seguido del elemento E2.

i_despues(2,[1,2,3],4,Lista)
Lista = [1,2,4,3]

i_despues(1,[1,2,3],2,Lista)
Lista = [1,2,2,3]

i_despues(1,Lista,2,[1,2,3,4])
Lista = [1,3,4]

Ejercicio 5.12 Haz un predicado **pos_elem(E1,L1)**, que escriba la posición del elemento E1 en la lista L1.

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

pos_elem(6,[1,2,3])
No

5.3.3 Cuando una posición es encontrada

El esquema general es:

/* Regla de terminación */

predicado(1,Cabeza, [Cabeza | Cola]):- procesar_algo.

/* Regla recursiva */

predicado(P, X, [_ | L]):- P1=P-1, predicado(P1,X, L).

Ejercicio 5.13 Escribe un predicado **posición(P,E1,L1)** que indique qué elemento E1 hay en la posición P de la lista L1.

posición(0, Elem, [1,2,6])
No Solution

posición(1,Elem,[4,5,6])
4

Ejercicio 5.14 Escribe un predicado **borra(N,L1,L2)** que sea verdad si la lista L2 es el resultado de borrar de la lista L1 el elemento que está en la posición N.

borra(1, Elem, [2,4,6], L)
L=[4,6]

borra(3,[2,4,6],L)
L = [2,4]

Ejercicio 5.15 Escribe un predicado **reem2(N,E,L1,L2)** que sea verdad si un elemento de la lista L1 en la posición N es reemplazado por E para dar la lista L2.

reem2(1, 4, [8,9,10], L)
L=[4,9,10]

reem2(3, 4, [8,9,10], L)
L=[8,9,4]

reem2(3, Elem, [8,9,10], [8,9,4])
Elem = 4

Ejercicio 5.16 Escribe un predicado **insertar2(N,E,L1,L2)** que sea verdad si L2 es la lista L1 con E precedido por el elemento que está en la posición N.

insertar2(1, 6, [9,10,11], L)
L=[6,9,10,11]

insertar2(3, 6, [9,10,11], L)
L=[9,10,6,11]

Ejercicio 5.17: El master_mind

master_mind([1,1,1,4,5], [1,1,7,5,3], M, H)
M=2, H=1

master_mind([1,2,3,4,5], [1,1,3,5,4], M, H)
M=2, H=2