Ing. de Sistemas e Informática
Matemáticas Discretas
Jorge M. Londoño P.

Taller de Prolog 7

Listas y Recursividad

Introducción a las listas

Las listas en Prolog se entienden como colecciones ordenadas de términos y se definen recursivamente de la siguiente forma:

- La lista vacía es una lista y se representa por [].
- Conocida una lista A y un termino t, se obtiene una nueva lista agregando t al comienzo de la lista. Esta operación se representa [t|A].
- No hay nada más que sea lista.

Asi por ejemplo, si a la lista vacía se le agrega el término c, la operación se representa [c|[]] y el resultado es la lista [c]. De igual manera, la operación [b|[c]] da por resultado [b,c] y la operación [a|[b,c]] da por resultado [a,b,c].

Observar que en la notación [t|A], el primer elemento siempre es un término (átomo, número, etc.) y el segundo elemento es una lista. Al primer término se le denomina la *cabeza de la lista* y a la lista A se le denomina la *cola de la lista*.

La unificación en listas opera unificando únicamente la cabeza y la cola de la lista. Por ejemplo la unificación [A|B] = [a,b,c] arroja por resultado el unificador A=a, B=[b,c]. Otros ejemplos:

- Unificar [A|B]=[a], arroja el unificador A=a, B=[] (porqué?)
- Unificar [A|B]=[] arroja falso (porqué?)

Algunos problemas simples de listas se resuelven por unificación, por ejemplo:

```
% primerElemento/2 : Encontrar el primer elemento de una lista
primerElemento(X, [X|_]).
% colaDeLista/2 : Determina el resto de la lista excluyendo la cabeza
colaDeLista(X, [_|X]).
% empiezanIgual/2 : Las dos listas empiezan por el mismo elemento?
empiezanIgual([X|_], [X|_]).
```

Hay muchas operaciones sobre listas que requieren considerar todos los elementos que componen la lista. En estos problemas, la solución del problema lleva a soluciones de forma recursiva. Consideremos algunos ejemplos:

1. Encontrar el último elemento de una lista

Si la lista tiene un solo elemento, este elemento es su último elemento (caso base). Si es

una lista de varios elementos, el último elemento es el último de la cola de la lista (caso recursivo). Observar que es falso que la lista vacía tenga último elemento.

```
% ultimoElemento/2 : Encuentra el último elemento de una lista
% arg1 : El último elemento
% arg2 : La lista
ultimoElemento(X, [X]) :- !.
ultimoElemento(X, [_|Cola]) :- ultimoElemento(X, Cola).
```

2. Determinar si la lista contiene un elemento

Si el elemento que queremos buscar es la cabeza de la lista, entonces si lo contiene (caso base). Sino, entonces buscamos el elemento en la cola de la lista (caso recursivo).

```
% miembroDeLista/2 : Determinar si arg1 es un elemento de la lista arg2
miembroDeLista(X, [X|_]) :- !.
miembroDeLista(X, [_|Tail]) :- miembroDeLista(X, Tail).
```

3. Sumar los elementos de una lista

La suma de los elementos de la lista vacía es cero (caso base). En los demás casos se suma la cabeza de la lista a la suma de los elementos de la cola (caso recursivo).

```
% sumaLista/2 : Suma los elementos de la lista de números arg1 y unifica el
resultado con arg2
sumaLista([], 0) :- !.
sumaLista([H|Cola], Total) :- sumaLista(Cola, Subtotal), Total is
H+Subtotal.
```

4. Imprimir los elementos pares de una lista

La lista vacía no tiene elementos pares (caso base). Si la cabeza de una lista no vacía tiene cabeza divisible por 2, entonces se imprime la cabeza y los pares del resto de la lista. Si no, entonces imprimir los pares en la cola de la lista.

```
imprimePares/1 : Imprime los números pares en una lista de números.
imprimePares([]) :- !.
imprimePares([H|T]) :- H mod 2 =:= 0, !, write(H), tab(2), imprimePares(T).
imprimePares([H|T]) :- imprimePares(T).
```

5. Encontrar el mayor elemento de una lista.

Se encuentra el menor elemento de la cola de la lista. Si es mayor que la cabeza de la lista, entonces este es el mayor. Sino, entonces el mayor es la cabeza de la lista. El caso base son las listas de un elemento, en cuyo caso este elemento es el mayor.

```
% maximo/2: Encuentra el mayor elemento de una lista
% arg1 : La lista
% arg2 : El mayor elemento de la lista
maximo([H], H).
maximo([H|T], M) :- maximo(T,M), M>H.
maximo([H|T], H) :- maximo(T,M), H>=M.
```

Una nota práctica:

Cuando una lista es larga, el Prolog la imprime en forma abreviada. Para imprimir la lista completa hay dos opciones[1]:

- Presionar **w** cuando se muestra el resultado de una consulta y el Prolog espera el Enter o el ';' (reintento).
- Cambiar la configuración por defecto para imprimir la lista completa. Esto se logra con la consulta:

```
set_prolog_flag(toplevel_print_options, [quoted(true), portray(true)]).
```

Ejercicios a desarrollar

1. Desarrollar un procedimiento `listaVacia/1` que tome una lista como arg1 y retorne verdadero si es la lista vacía. En cualquier otro caso debe retornar falso. Por ejemplo, `listaVacia([4,2,9]) returna false`.

Realizar las siguientes consultas:

- Consultar `listaVacia` de una lista con 10 números.
- Consultar `listaVacia` de una lista con 5 elementos, que combine números y átomos.
- Consultar `listaVacia` de una lista con 1 elemento.
- Consultar `listaVacia` de la lista vacía.
- Consultar `listaVacia` con un argumento que no sea una lista.

2. Desarrollar un procedimiento `contarNumeros/2` para determinar cuántos números hay en una lista. Por ejemplo `contarNumeros([a,2,7,x],N)` debe retornar N=2. Sugerencia: Hacer uso del predicado `number/1` que determina si el argumento es un número.

Realizar las siguientes consultas:

- Contar números de la lista [9,7,5,3].
- Contar números de la lista [9,a,7,b,5,c,3,d].
- Contar números de la lista [a,b,c,d,e].
- Contar números de la lista vacía. Es correcto el resultado obtenido?

1 http://www.swi-prolog.org/FAQ/AllOutput.html

3. Desarrollar un procedimiento minimo/2 que toma 2 argumentos: Una lista de números y una variable para el menor elemento. Al finalizar, el procedimiento retorna el menor valor que aparezca en la lista. Asi por ejemplo, `minimo([3,1,5,9,4], Menor)` returna Menor=1.

Realizar las siguientes consultas:

- Menor de una lista de 10 números.
- Menor de una lista con números y átomos.
- Menor de una lista con un elemento.
- Menor de la lista vacía.

4. Desarrollar un procedimiento menorYmayor/3 que toma como argumentos una lista de números y dos variables. El procedimiento devuelve en las variables los valores menor y mayor que aparezcan en la lista.

Realizar las mismas consultas utilizadas en el numeral 3.

Informe

Implementar sus procedimientos en una base de datos <NombreApellido>-<id>-`bd7.pl` y las consultas respectivas en un archivo de texto <NombreApellido>-<id>-`consultas7.txt`. Documentar apropiadamente su programa.