

# Programación Declarativa, 2023-2

## Nota Adicional sobre Listas diferencia en PROLOG

Luis Fernando Loyola Cruz

Manuel Soto Romero

13 de febrero de 2023  
Facultad de Ciencias UNAM

En teoría de conjuntos, la diferencia es una operación que da como resultado otro conjunto cuyos elementos son todos aquellos elementos del primer conjunto que no se encuentren en el segundo. La diferencia de conjuntos usualmente es representada con el operador infijo  $-$  de manera que, para representar la diferencia de dos conjuntos escribimos  $A-B$  en donde  $A$  y  $B$  son conjuntos.

Esta noción de *diferencia* se puede aplicar a otras estructuras de datos, en particular a las listas. Por ejemplo, consideremos la lista  $[a,b,c]$ . Esta lista la podemos representar de diferentes maneras:

- $[a,b,c]-[]$
- $[a,b,c,d,e]-[d,e]$
- $[a,b,c|X]-X$

La Figura 1 muestra un diagrama con la primera forma de representación.

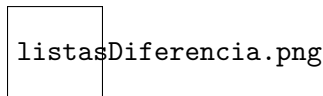


Figura 1: Representación gráfica de las listas diferencia.[1]

Es importante recordar que internamente las listas son construidas con el término compuesto `cons`, que recibe dos argumentos: un elemento y otra lista. De esta forma podemos representar la lista  $[1,2,3]$  como:

```
cons(1, cons(2, cons(3, [])))
```

Por otro lado, la lista  $[1,2|X]$  se representa como:

```
cons(1, cons(2, X))
```

en donde  $X$  es una variable.

Dado que no conocemos el valor de  $X$ , decimos que la lista  $[1,2|X]$  es una *lista abierta* y se dice (informalmente) que  $X$  es un *hueco*. Sin embargo, ¿qué sucede si unificamos  $X$  con una lista  $[3]$ ?

```
?- Lista = [1,2|X], X = [3]
Lista = [1,2,3]
```

Hemos *llenado* el hueco y ahora tenemos una lista *ordinaria*.

Generalmente cuando queremos en realizar una operación sobre una lista pensamos en tomar una lista ordinaria como entrada y devolver una lista ordinaria como salida, sin embargo, no hay nada que nos impida representar listas como listas abiertas, de esta forma podemos pensar en tomar listas abiertas como entrada y devolver listas abiertas como salida. Ciertamente, podemos convertir una lista abierta en una lista ordinaria simplemente *llenando* el hueco, por ejemplo:

```
?- Lista = [a,b,c|X], X = [d,e]
Lista = [a,b,c,d,e]
```

Un momento, ¿acaso acabamos de concatenar dos listas como lo haría el predicado `append/3`?  
¿Qué sucedería si en vez de tomar listas ordinarias como entrada permitiéramos recibir listas abiertas?

Definamos el predicado `concatena1/4` de la siguiente manera :

```
concatena1(A,B,B,A).
```

Ahora hagamos un pequeño experimento:

```
concatena1([a,b,c|X],X,[d,e],Lista).
X = [d,e]
Lista = [a,b,c,d,e]
```

¡Parece que nuestro predicado para concatenar listas funciona!

Veamos como:

- A se unifica con `[a,b,c|X]`.
- B se unifica con X
- B se instancia a `[d,e]`
- A se unifica con Z

¿Y si usamos listas diferencia para definir el predicado? Definamos el predicado `concatena2/3` de la siguiente manera:

```
concatena2(A-B, B, A).

?- concatena2([a,b,c|X]-X, [d,e], Lista).
X = [d,e]
Lista = [a,b,c,d,e]
```

Teniendo en cuenta que estamos trabajando con listas diferencia y listas abiertas lo ideal sería recibir como entrada listas diferencia y regresar una lista diferencia, por lo que redefiniremos nuestro predicado obteniendo:

```
concatena3(A-B, B-C, A-C).

?- concatena3([a,b,c|X]-X, [d,e|Y]-Y, Z1-Z2).
X = [d,e|_G123]-_G123
Y = _G123
Z1 = [a,b,c,d,e|_G123]-_G123
```

en donde `_G123` es un *hueco*.

La Figura 2 muestra cómo funciona nuestra regla.

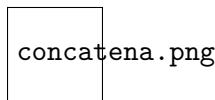


Figura 2: Representación gráfica de la concatenación usando listas diferencia[1].

Usando el predicado predefinido `listing/1` podemos ver como esta implementado el predicado `append/3`:

```
?- listing(append/3).
append([], A, A).
append([A|B], C, [A|D]) :- append(B, C, D).
```

Podemos observar como la complejidad en tiempo es linealmente proporcional a la longitud del primer argumento. Sin embargo, con el predicado `concatena3/3` que acabamos de definir, la complejidad de concatenar listas es constante.

## Reversa de una lista

Ahora que sabemos concatenar listas en tiempo constante veamos cómo podemos optimizar otros predicados sobre listas.

La definición usual del predicado predefinido `reverse/2` es la siguiente:

```
?- listing(reverse/3).
reverse([], []). % clause
reverse([H|T], R) :-
    reverse(T, L),
    append(L, [H], R).
```

La Figura 3 muestra cómo calcula la reversa el predicado `reverse`.


reversa.png

Figura 3: Representación gráfica del predicado `reversa`[1].

Como ya sabemos el predicado `append/3` es bastante ineficiente. Redefinamos el predicado `reversa/2` usando listas diferencia.

```
reversa([], L-L).  
reversa([H|T], L1-L2) :- reversa(T, L1-[H|L2]).
```


reversaLD.png

Figura 4: Interpretación gráfica del predicado `reversa/2` usando listas diferencia[1].

Lo anterior toma la siguiente interpretación:

*La lista diferencia  $L1-L2$  es la reversa de  $[H|T]$  si la lista diferencia  $L1-[H|L2]$  es la reversa de  $T$ .*

Aquí podemos apreciar como es que las listas diferencia en PROLOG pueden ser vistas como la diferencia de listas.

## Referencias

- [1] Attila Csenki, *Difference lists in Prolog*, Teaching Mathematics for Computer Science, 2010.