Practica 3: PROLOG

Tomas Higuera Viso y Manuel Chamorro Martinez de Aragon

1) Implemente un predicado duplica(L,L1), que es cierto si la lista L1 contiene los elementos de L duplicados.

Para resolver este ejercicio hemos creado un predicado que tiene como caso base ambas listas vacias y que si no lo estan comprueba si los dos primeros elementos de la lista L1 son igual al primero elemento de L. Si lo son realizamos una recursion con el resto de elementos de la lista.

Casos de prueba:

- Prueba 1: duplica([1, 2, 3], [1, 1, 2, 2, 3, 3]).
  Resultado: true

- Prueba 2: duplica([1, 2, 3], [1, 1, 2, 3, 3]).
  Resultado: false

- Prueba 3: duplica([1, 2, 3], L1).
  Resultado: L1 = [1, 1, 2, 2, 3, 3]

- Prueba 4: duplica(L, [1, 2, 3]).
  Resultado: false

Todos los resultados se corresponden a los especificados en el enuciado.

2) Implementa el predicado invierte(L, R) que se satisface cuando R contiene los elementos de L en orden inverso.

Para resolver este ejercicio hemos creado un predicado cuyo caso base son ambas listas vacias. La recursion que realizams es la comprobacion de si el primer elemento de la lista se corresponde con el resultado de concantenerlo al final de la lista, si asi es continuamos con la recursion del resto de elementos.

Casos de prueba:

- Prueba 1: concatena([], [1, 2, 3], L).
  Resultado: L = [1, 2, 3]

- Prueba 2: concatena([1, 2, 3], [4, 5], L).
  Resultado: L = [1, 2, 3, 4, 5]

- Prueba 3: invierte([1, 2], L).
  Resultado: L = [2, 1]

- Prueba 4: invierte([], L).
  Resultado: L = []

Todos los resultados se corresponden a los especificados en el enuciado.

Tomas Higuera Viso y Manuel Chamorro Martinez de Aragon

3) Implementar el predicado palindromo(L) que se satisface cuando L es una lista palíndroma, es decir, que se lee de la misma manera de izquierda a derecha y de derecha a izquierda.

Para resolver este ejercicio hemos realizado un predicado que comprueba si la lista es igual a su lista invertida haciendo uso del predicado invierte.

Casos de prueba:

- Prueba 1: palindromo([1, 2, 1]).
  Resultado: true

- Prueba 2: palindromo([1, 2, 1, 1]).
  Resultado: false

Todos los resultados se corresponden a los especificados en el enuciado.

- ¿Qué pasa si se llama palindromo(L), donde L es una variable no instanciada?

    Si llamamos al predicado palindromosin instanciar obtenemos un bucle infinito, ya que prolog interpreta tambien como elementos las posiciones de memoria, por lo que obtenemos palindromos de posiciones de memoria.

4) Implementar el predicado divide(L,N,L1,L2) que se satisface cuando la lista L1 contiene los primeros N elementos de L y L2 contiene el resto.

Para realizar este predicado hemos creado dos posibles casos, uno en el que nos dan una lista vacia y el numero de elementos en los que hay que dividirla es 0, por lo que el predicado se satisface. El otro caso posible es que la lista no este vacia por lo que haciendo uso de la funcion length creamos otra lista que contiene los N primeros elementos de L y comprobamos si la concatenacion de L1 y L2 dan como resultado L.

Casos de prueba:

- Prueba 1: divide([1, 2, 3, 4, 5], 3, L1, L2).
  Resultado: L1 = [1, 2, 3]
  L2 = [4, 5]

- Prueba 2: divide(L, 3, [1, 2, 3], [4, 5, 6]).
  Resultado: L = [1, 2, 3, 4, 5, 6]

Todos los resultados se corresponden a los especificados en el enuciado.

5) Implementar el predicado aplasta(L, L1) que se satisface cuando la lista L1 es una versión "aplastada" de la lista L, es decir, si uno de los elementos de L es una lista, esta será remplazada por sus elemento, y así sucesivamente.

Para resolver este ejercico hemos creado un predicado cuyo caso base son dos listas vacias. Si ambas listas no son vacias hemos evaluado dos posibles casos. En el primero comprobamos si el primer elemento de la lista es otra lista y si lo es llamamos recursivamente al predicado aplasta con el primer elemento de la lista y con el resto de la lista. El resultado de ambas llamadas es concatenado y guardado en L1. Si el primer elemento de la lista no es una lista lo guardamos en L1 y hacemos la llamada recursiva a aplasta con el resto de elementos.

Casos de prueba:

- Prueba 1: aplasta([1, [2, [3, 4], 5], [6, 7]], L).
  Resultado: L = [1, 2, 3, 4, 5, 6, 7]

- Prueba 2: aplasta(L,[1, 2, 3]).
  Resultado: L = [1, 2, 3]
  El resultado obtenido al realizar esta ejecucion es la misma lista, ya que al hacer la ejecucion de manera inversa guarda los elementos igual que los del resultado sin crear otras posibles interpretaciones.

Todos los resultados se corresponden a los especificados en el enuciado.

6) Implementar el predicado primos(N, L) que se satisface cuando la lista L contiene los factores primos del número N.

Para realizar este ejercicio hemos creado una funcion auxiliar next_factor diferente a la propuesta en el enunciado. Esta funcion va comprobando numero a numero si el elemento N es divisible y si lo es lo guarda como factor y sigue comprobando el resto de facotores con el resultado de la division de N y el numero. Hemos utilizado esta implementacion ya que es mas general que la propuesta en el enunciado pero menos optima, ya que al realizar la comprobacion de todos los numeros siempre comprobamos los que son divisbles por dos y por tanto no son factores primos.
Hemos conseguido resolver este ejercicio sin poder concatenar el resultado completo en una lista, por lo que los factores se van devolviendo uno a uno.

Casos de prueba:

- Prueba 1: primos(100,L).
  Resultado: 2
  2
  5
  5

El resultado se corresponde con el del enunciado con los factores primos sin ser almacenados en una lista.

7.1) Empezamos con el predicado cod_primero(X, L, Lrem, Lfront). Lfront contiene todas las copias de X que se encuentran al comienzo de L, incluso X; Lrem es la lista de elementos que quedan.

Para realizar este ejercicio hemos creado un predicado con tres posibles casos. El primero de ellos contempla que tras la recursion el ultimo elemento de la lista se corresponde con el que buscamos, por lo que lo guardamos en Lfront y Lrem lo dejamos como una lista vacia. El segundo caso realiza la recursion que va almacenando todos los elementos iguales que buscamos en Lfront. El ultimo caso guarda el resto de elementos en Lrem cuando estos dejan de ser iguales al elemento X que buscamos.

Casos de prueba:

- Prueba 1: cod_primero(1, [1, 1, 2, 3], Lrem, Lfront).
  Resultado: Lfront = [1, 1, 1]
  Lrem = [2, 3]

- Prueba 2: cod_primero(1, [2, 3, 4], Lrem, Lfront).
  Resultado: Lfront = [1]
  Lrem = [2, 3, 4]

Todos los resultados se corresponden a los especificados en el enuciado.

7.2) El predicado cod_all(L, L1) aplica el predicado cod_primero/4 a toda la lista L.

Para realizar este ejercicio hemos creado un predicado cuyo caso base son ambas listas vacias. Si ambas listas no estan vacias se realiza una recursion haciendo uso de cod_primero en la que se almacenan todos los elementos de la lista L que son iguales en una lista, que a su vez se encuentra en la lista de listas L1. Para realizar correctamente este predicado hemos tenido que quitar uno de los elementos del resultado de cod_primero, ya que ademas de los elementos iguales de la lista tambien almacenabamos el elemento que buscabamos, es decir, un elemento extra.

Casos de prueba:

- Prueba 1: cod_all([1, 1, 2, 3, 3, 3, 3], L).
  Resultado: L = [[1, 1], [2], [3, 3, 3, 3]]

Todos los resultados se corresponden a los especificados en el enuciado.

7.3) El predicado run_length(L, L1) aplica el predicado cod_all y luego transforma cada una de las listas en la codificación run-length.

Para realizar este ejercicio hemos creado dos predicados auxiliares ademas del solicitado. Uno de ellos es el predicado first que devuelve el primer elemento de una lista y el otro es el predicado contador_elementos que hace una recursion guardando de cada elemento de tipo lista el primer elemento de la misma, asi como el numero de elementos que la componen.
En la funcion run_length hacemos uso del predicado cod_all asi como el de contador_elementos, por lo que no tenemos que realizar ninguna recursion.

Casos de prueba:

- Prueba 1: run_length([1, 1, 1, 1, 2, 3, 3, 4, 4, 4, 4, 4, 5, 5], L).
  Resultado: L = [[4, 1], [1, 2], [2, 3], [5, 4], [2, 5]]

Todos los resultados se corresponden a los especificados en el enuciado.