COL 765 - ILFP Assignment 1

Submitted By : Om Prakash (2019MCS2567)

1 Definitions

1.1 Ocaml

OCaml is a general-purpose, multi-paradigm programming language which extends the Caml dialect of ML with object-oriented features.

1.2 Sets

A set is a well-defined collection of distinct objects. The arrangement of the objects in the set does not matter.

1.3 Lists

An OCaml list is an immutable, finite sequence of elements of the same type.

2 Operations on Lists

2.1 set

This functions takes a list as an input, removes duplicate element(s) if any exists and returns a list with distinct elements as a set has always distinct elements.
Input: A list with or without duplicate elements
Output: list with distinct elements
Time Complexity : $O(n^2)$

2.2 emptyset

If the List is empty this function returns true otherwise if the list has at least single element it will return false.
Input: A list
Output: true or false
Time Complexity : $O(1)$

2.3 member

This function recursively removes head from the list and matches it with the given element. If the match found it returns true and if the given list becomes empty after complete processing it returns false.
Input: A list, one element
Output: true or false
Time Complexity : $O(n)$

2.4 intersection

This function takes two lists as input and find their intersection. We take elements from the first list one by one and check if it is present in the second list, if yes, we keep it in the result else ignore it. If any of the list is empty it returns empty list without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: A list
Time Complexity : $O(m * n)$ where m is the length of the first list and n is the length of the second list.

2.5 union

This function takes two lists as input and find their union. We take elements from the first list one by one and check if it is present in the second list, if not, we add it to the second list else ignore it, and finally returns the second list. If any of the list is empty it returns the other list as input without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: A list
Time Complexity : $O(m * n)$

2.6 difference

This function takes two lists as input and find their $A - B$. It takes elements one by one from the first list and checks if it is present in the second list, **if not**, keep it in the result else ignore it. If the first list is empty it returns empty list and if the second list is empty it returns the first list as it as without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: A list
Time Complexity : $O(m * n)$

2.7 equal

This function takes two lists as input and find their differences $A - B$ and $B - A$ if both the results are empty then both the given lists are equal. If one of the list is empty and the other list is non-empty it returns false and if both the lists are empty then it return true without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: true or false
Time Complexity : $O(m * n)$

2.8 subset

This function takes two lists as input and find if the first list is a subset of second list. It checks if every element of the first list is a member of the second list, if yes then it returns true otherwise false. If the first list is empty it return true and If the first list is non-empty and the second list is empty it returns false without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: true or false
Time Complexity : $O(m * n)$

2.9 product

This function takes two lists as input and find if the cross products of these two lists. It takes elements one by one from the first list and multiply it with every elements of the second list. If any of the list is empty it returns empty list without doing any further processing hence its best case is $O(1)$.
Input: Two lists
Output: cross product list
Time Complexity : $O(m * n)$

2.10 power

This function takes a list and returns a list of all its sub-lists. First, it removes the head element and solve recursively for the remaining elements. When the recursive call returns, we combine this intermediate result with the head element we previously removed from the list and return each of the sub-list with the head element added in front of it.
Input: Single list
Output: A list which contains all possible sub-lists
Time Complexity : $O(2^n)$

3 Assignment 1 (Part b)

In mathematics, an indicator function or a characteristic function is a function defined on a set X that indicates membership of an element in a subset A of X, having the value 1 for all elements of A and the value 0 for all elements of X not in A.

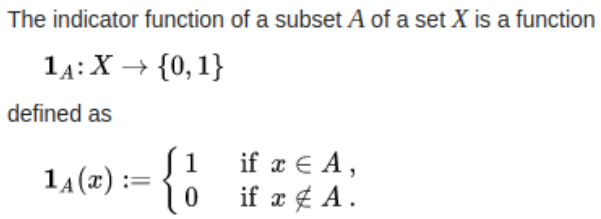

Figure 1: Indicator function (*reference: Wikipedia*)

3.1 empty set

let emptyset () = false;;

As per the above mentioned definition of a characteristics function, *a function defined on a set X that indicates membership of an element in a subset A of X, having the value 1 for all elements of A and the value 0 for all elements of X not in A.*
As empty set has no element hence it's not possible to find its member(s).

3.2 member

let member x f = f x;;

if x is a member of f then it will return true else false which itself is the property of a characteristics function of membership.

3.2.1 testing functions

1. let even x = if x mod 2 = 0 then true else false;;
2. let m20 x = if x mod 20 = 0 then true else false;;

3.3 union

let union f g x = f x || g x;;

Union operation will return true if x is a member of function f or function g or both otherwise it will return false.

3.4 intersection

let intersection f g x = (f x) && (g x);;

Intersection operation will return true if x is a member of function f **and** function g, otherwise it will return false.

3.5 difference

let difference f g x= (f x) $& (not(g x));;

If difference operator is applied on two sets A and B, the it returns the elements which are member of A but not the member of B. Therefore, it will return true if x is member of A and not member of B.

3.6 product

let product f g (x,y) = (f x) && (g y);;

In the product operation, every element of the first set is multiplied with all the elements from the second set. Therefore, it will return true if x belongs to f and g belongs to y, therefore product f g (x,y) returns true if x belongs to f and y belongs to y.