CSCI 4220
– Assignment 4 –

1 Instructions

An important aspect of programs written in a functional programming language like SML is that recursion is used extensively. The purpose of this assignment is to give you some practice expressing your computational thoughts using recursion. Also, the last 2 problems in this assignment are similar to the *access* and *update* functions you will need to write for Milestone 4.

You must submit your solutions to this assignment electronically to Canvas. Your submission should consist of the following.

- A single pdf file containing all your solutions. This will be the file that I will electronically mark-up and grade.
- One or more dot-sml files containing executable code for every problem solution.

Note that when solving a particular problem, you may freely make use of any auxiliary functions you deem necessary. So, for example, a problem solution can consist of one main function that makes calls to one or more auxiliary functions.

Important: You should avoid solutions to problems that *reuse* functions in an unnatural or inefficient manner.

1.1 Notes on Grading:

1. A correct but disorganized solution receives a grade of C
2. A correct and relatively organized solution receives a grade of B
3. A correct and elegant solution receives a grade of A

1.2 Advice:

1. Don't start coding before you fully understand the problem.
2. After you get a correctly running program consider the problem halfway completed. Spend a reasonable amount of effort cleaning up the solution and looking for better solutions.

2 Basic List Manipulations

Problem 1 *(5 points) Write an ML function, called* alternate: *'a list * 'a list → 'a list, that takes two lists, having the same length, as input and produces an output list whose elements are alternately taken from the first and second input list respectively.*

Example: alternate([1,3,5],[2,4,6]) = [1,2,3,4,5,6]

Problem 2 *(5 points) Write an ML function, called* minus*: int list * int list → int list, that takes two non-decreasing integer lists and produces a non-decreasing integer list obtained by removing the elements from the first input list which are also found in the second input list. For example,*

minus([1,1,1,2,2], [1,1,2,3]) = [1,2]
minus([1,1,2,3],[1,1,1,2,2]) = [3]

Problem 3 *(20 points) Write a function, called* union*: ''a list * ''a list → ''a list, that when called with two lists, denoting sets, returns their union. In other words,*

union($s1$,$s2$) = $s1 \cup s2$ where $s1$ and $s2$ are lists.

Recall that sets may not contain duplicate elements. (Hence $s1 \cup s2$ may not contain duplicate elements).

Problem 4 *(20 points) Write a function, called* multiSetIntersection*: ''a list list → ''a list, that when passed a list of sets (which are also represented as lists) as input will return their intersection as output. For example,*

multiSetIntersection([$s1$,$s2$,$s3$,$s4$]) = $s1 \cap s2 \cap s3 \cap s4$
multiSetIntersection([]) = []

Remark. An intersection of two sets is the set of all elements that are common to both sets. That is, an element x is a member of the intersection of $S1$ and $S2$, if and only if x is a member of $S1$ and x is a member of $S2$. This can be formally expressed as follows:

$$x \in (S1 \cap S2) \Leftrightarrow x \in S1 \wedge x \in S2$$

Problem 5 *(10 points) The Cartesian product (or cross product) of two sets, $S1$ and $S2$, is often denoted by the symbol $\times$ and is defined follows: $S1 \times S2 = \{(x, y) \mid x \in S1 \wedge y \in S2\}$. Using tuple and list notation, write a function called* crossProduct*: 'a list * 'b list → ('a * 'b) list that takes two lists as input and returns their Cartesian product.*

Problem 6 *(20 points) The powerset of a set S is defined as the set of all subsets of S (including the empty set). For example, the powerset of $\{1, 2\}$ is $\{\{1, 2\}, \{1\}, \{2\}, \{\}\}$. Write an SML function called* powerset*: 'a list →' a list list that when given a list representation of a set as input returns its powerset as output.*

3 Representing Functions as Sets (or Lists)

Any function from integers to integers can be expressed as a **set** of input-output pairs. Consider the function *posIntegerSquare*: int $\rightarrow$ int, that takes a positive integer[1], x, as input and returns $x * x$.

Function represented as a computation. fun posIntegerSquare x = x*x
Function represented as a set. val posIntegerSquare = [(1,1), (2,4), (3,9),...]

The advantage of set representation is that, aside from the lookup operation, no calculations need to be performed in order to obtain the output value of the function. On the other hand, the disadvantage of the set representation is that, for most functions, the set of input-output pairs will be infinite. Thus, constructing such a list is not feasible.

Problem 7 *(10 points) Write an SML function, called* ***finiteListRepresentation****:* $(int \rightarrow\ 'a) * int \rightarrow (int * 'a)\ list$*, that takes as input an arbitrary function* f*:* $int \rightarrow\ 'a$*, and a positive integer,* n*, and returns the list representation of* f *corresponding to the first* n *input-output pairs.*

Example. finiteListRepresentation(posIntegerSquare, 5) = [(1,1), (2,4), (3,9), (4,16), (5,25)]

Remark. Note that in this problem, the output list denotes a set. Also note that in a set the order of elements is not important.

Problem 8 *(10 points) Write an SML function, called* ***update****:* $(''a * 'b)\ list * (''a * 'b) \rightarrow (''a * 'b)\ list$*, that takes a finite list representation of a function as a list of input-output pairs and returns an updated finite list representation. For example, let* $(x, f(x))$ *denote an arbitrary input-output pair, and let* $FLR = [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$ *denote an arbitrary finite list representation. If there exists a value for* i *such that* $1 \leq i \leq n \wedge x_i = x$*, then update(*$FLR, (x, f(x))$*) will cause the element* (x_i, y_i) *in* FLR *to be replaced with* $(x, f(x))$*. On the other hand, if there does not exist a value for* i *such that* $1 \leq i \leq n \wedge x_i = x$*, then the element* $(x, f(x))$ *should be added to* FLR*.*

Let FLR = [(1,1),(2,4),(3,9),(4,16),(5,25)].

1. update(FLR, (2,3)) = [(1,1),(2,3),(3,9),(4,16),(5,25)]
2. update(FLR, (6,36)) = [(1,1),(2,4),(3,9),(4,16),(5,25),(6,36)]

[1]This is a precondition that is assumed but not explicitly enforced.