Homework 6: High Order ML Functions

Stephen Wagstaff
CS 431
October 18, 2018

---

1. Write a function `reduce: ('a * 'a -> 'a) -> a' list -> 'a list` that behaves like `foldl` except that it takes the first element of the list as the initial value. For example, `reduce (op -) [1,2,3]` evaluates to `3 - (2 - 1) = 2`. This method does not apply to empty list.

```
fun reduce _ nil = raise Fail "Empty list passed."
  | reduce _ (first::nil) = first
  | reduce f (first::rest) = f(first, reduce f rest);
```

For the rest of the questions, use `zip`, `map`, and `reduce` to solve the problems, where `zip` function is from homework 4. The following questions are about vectors and matrix. We represent row vectors using lists. For example, `[2,3,5,4]` represents a row vector of four integers. We represent a matrix using a list of lists. For example, the matrix

```
1 2 3
4 5 6
```

is written as `[ [1,2,3], [4, 5, 6] ]`.

---

2. Write a function `vectorAdd: int list * int list -> int list` that add two integer vectors of the same size.
   - For example, `vectorAdd ([1,2,3], [4,5,6])`should return `[5, 7, 9]`.

```
fun vectorAdd(nil, nil) = nil
  | vectorAdd(nil, l2) = l2
  | vectorAdd(l1, nil) = l1
  | vectorAdd(l1, l2) = map (fn (x, y) => x + y) (zip(l1,l2));
```

3. Write a function `svProduct: int * int list -> int list` that multiple an integer with an integer list.

- For example, `svProduct(2, [1,2,3])` should return `[2,4,6]`.

```
fun svProduct (_, nil) = nil
  | svProduct (mult, list) = map (fn y => mult * y) list;
```

4. Write a function `vmProduct` that multiple a row vector of size `n` with a matrix with `n` rows and `m` columns to produce a vector of size `m`.

- For example, `vmProduct([1,2,3], [[1,1], [2,1], [3,1]])` should return `[14, 6]`. Or,

```
              1    1
[1  2  3]× 2    1 = 1×[1  1] + 2×[2  1] + 3×[3  1] = [1  1] + [4  2] + [9  3]= [14  6]
              3    1
```

This function uses the functions `svProduct` and `vectorAdd` defined earlier.

```
fun vmProduct(rowVector, matrix) =
  reduce vectorAdd  (map (fn (x, list) => svProduct(x, list)) (zip(rowVector, matrix)));
```

5. Write a function `matrixProduct` that multiple a `m×n` matrix with a `n×k` matrix to obtain a `m×k` matrix.
   - For example

```
 1    2    3   1    1
 1    1    1 x 2    1 = v1 = 14 6
               3    1   v2 = 6  3

  where
                    1    1
  v1 = [1  2  3] × 2    1 = [14    6]
                    3    1
```

```
    and
                    1    1
    v2= [1  1  1] × 2    1 = [6    3]
                    3    1
    That is,
    matrixProduct([ [1, 2, 3], [1, 1, 1] ], [ [1, 1], [2, 1], [3, 1] ])
      = [ [14, 6], [6, 3] ]
```

This problem will use the function `vmProduct` defined previously.

```
fun matrixProduct(matrix1, matrix2) =
  map (fn row => vmProduct(row, matrix2)) matrix1;
```