Structure and Interpretation of Computer Programs

R7A
Problem Solving and List Processing

LISTS Functions of Source §2

The LISTS functions are a set of predeclared functions provided in Source §2 for list processing. Please refer to the the online reference for details of these LISTS functions.

Problems:

1. Write the function `map` **using `accumulate`**. Name your function `my_map`.

   ```
   function my_map(f, xs) {
       // this should be a one-liner

   }
   ```

   Example calls:

   ```
   my_map(x => x + 1, list(1, 2, 3));
   // Result: list(2, 3, 4)
   ```

2. Write a function called `remove_duplicates` that takes in a list as its only argument and returns a list with duplicate elements removed. The order of the elements in the returned list does not matter. **Use `filter` in your function.**

   ```
   function remove_duplicates(lst) {
       ...
   }
   ```

   Example calls:

   ```
   remove_duplicates(list(1, 2, 3, 4, 4, 3, 2, 1, 2));
   // Result: list(1, 2, 3, 4)

   remove_duplicates(list("a", "x", "b", "c", "c", "b", "d"));
   // Result: list("a", "x", "b", "c", "d")
   ```

3. Our friend Louis Reasoner has a pocket full of change. He wants to buy a snack that will cost him x cents, and he wants to know all the ways in which he can use his change to make up that amount. Please help him in writing a function which takes as parameters the amount `x` and a list of all the coins Louis has in his pocket, and returns a list of lists, such that each sub-list of the result contains a valid combination to make up `x`. A combination may appear more than once, since it may be using different coins of the same denomination.

   Help Louis by filling in the ellipses ... in his incomplete solution:

```
function makeup_amount(x, coins) {
    if (x === 0) {
        return list(null);
    } else if (x < 0 || is_null(coins)) {
        return null;
    } else {
        // Combinations that do not use the head coin.
        const combi_A = ...

        // Combinations that do not use the head coin
        // for the remaining amount.
        const combi_B = ...

        // Combinations that use the head coin.
        const combi_C = ...

        return append(combi_A, combi_C);
    }
}
```

Example call:

```
makeup_amount(22, list(1, 10, 5, 20, 1, 5, 1, 50));
// Result: list(list(20, 1, 1), list(10, 5, 1, 5, 1), list(1, 20, 1),
//              list(1, 20, 1), list(1, 10, 5, 5, 1),
//              list(1, 10, 5, 1, 5))
```

Note: The sublist `list(1, 20, 1)` appears twice. Each appearance of the number `1` refers to a different coin.