National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2016/2017

Recitation 5
Lists and List Processing I

The Source

1. *v1*===*v2* — returns `true` if *v1* is identical to *v2*. This means they are exactly the same in case of numbers, boolean values or strings, or come from the same (function, pair, empty list) creation, in case of function values, pairs and empty lists.
2. `equal(`*v1*`,` *v2*`)` — returns `true` if *v1* and *v2* enjoy structural equality. This is the case, if they are both the empty list. If they are both pairs, their head and tail need to enjoy structural equality. In all other cases, they must be identical (===).
3. `member(`*object*`,` *list*`)` — returns the first tail of *list* whose head is identical (===) to *object*, or returns `[]` if *object* does not occur in *list*.

Problems:

1. Give printed values of

```
equal(1, 1)

equal(0, 1)

equal("foo", "foo")

equal("foo", "bar")

equal(0, "0")

equal(false, false)

equal(false, "false")

equal(pair(1, 2) , pair(1, "2"))

equal(pair(1, 2) , pair(1, 2))

equal(list(1, 2, 3, 4, 5) , list(1, 2, 3, 4, 5))

equal(list(list(1,2), list(2,3), list(4,5)), list(list(1,2), list(2,3), list(4,5)))

equal(list(list(1,2), list(2,3), list(4,5)), list(list(1,2), list(2,3)))
equal(list(list(1,2), list(2,3)), list(list(1,2), list(3,2)))

equal([], [])

equal([], list(1))

equal(list(1), pair(1,[]))
```

2. **[SICP Ex 2.38]** The `accumulate` function is also known as `fold_right`, because it combines the first element of the sequence with the result of combining all the elements to the right.

   ```
   function accumulate(op, initial, sequence) {
       if(is_empty_list(sequence)) {
           return initial;
       } else {
           return op(head(sequence), accumulate(op, initial, tail(sequence)));
       }
   }
   var fold_right = accumulate;
   ```

   There is also a `fold_left`, which is similar to `fold_right`, except that it combines elements by working in the opposite direction:

   ```
   function fold_left(op, initial, sequence) {
       function iter(result, rest) {
           if(is_empty_list(rest)) {
               return result;
           } else {
               return iter(op(result, head(rest)), tail(rest));
           }
       }
       return iter(initial, sequence);
   }
   ```

   Given the following function:

   ```
   function div(x, y) {
       return x / y;
   }
   ```

   What are the values of

   ```
   fold_right(div, 1, list(1,2,3))
   fold_right(pair, [], list(1,2,3))
   fold_left(div, 1, list(1,2,3))
   fold_left(pair, [], list(1,2,3))
   ```

   Give a property that `op` should satisfy to guarantee that `fold_right` and `fold_left` will produce the same values for any sequence.

3. This question asks you to build a variant of the solution to the “Towers of Hanoi” problem presented in class. We define a *disk move* to be a **list** of two numbers: the source pole and the destination pole. For example, `[1, [3, []]]` indicates the move of a disk from the first pole to the third.

Write a function called `hanoi`, which takes in 4 parameters:

- the number of disks,
- the source pole.
- the destination pole,
- the auxiliary pole,

and *returns a **list of disk moves*** that, if executed in that sequence, will move all the disks from the source pole to the destination pole and comply with the rules of the Towers of Hanoi game. You may make use of the `append` function, provided in Source Week 5 and later.

```
function append(list1, list2) {
   if (is_empty_list(list1)) {
      return list2;
   } else {
       return pair(head(list1), append(tail(list1), list2));
   }
}
```

Example call:

```
hanoi(3, 1, 2, 3);
// Returned value:
//    list(list(1, 2), list(1, 3), list(2, 3), list(1, 2),
//         list(3, 1), list(3, 2), list(1, 2))
```