CSE 461: Programming Languages Concepts

Prof. G. Tan
Spring 2022

Homework 6: Memoization in Racket
Due on Apr 20th at 6pm in Canvas.

Submission: you should submit via Canvas a DrRacket file with Racket code in. Please clearly mark your answers using comments so that we can tell the correspondence between your code and questions.

1. (5 points) In this task, we define a function called `ackermann_mem`, which is the memoized version of the Ackermann function. The mathematical definition of the Ackermann function is as follows:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

   - Define the Ackermann function called `ackermann` in Racket.
   - Define the `bind` and `lookup` functions for association lists, as we discussed in class. Recall that an association list in Racket is just a list of pairs and each pair contains a key and a value.
     - `(bind k v al)` returns a new association list, which is the result of adding a new entry `(k,v)` to the beginning of association list `al`.
     - `(lookup k al)` returns the value for key `k` in `al` if there is an entry for `k` and returns `#f` otherwise.
   - Define a global variable `al` for the association list used in `ackermann_mem`.

```
(define al '())
```

   - Finally, define `ackermann_mem`. When given `m` and `n`, it checks whether there is an entry for key `(m n)` in `al`; note this association list maps a pair `(m n)` to the result of `(ackermann m n)`. If there is, it returns the value in the entry; if not, it invokes `(ackermann m n)`, adds the entry `((m n) (ackermann m n))` to the association list, and returns `(ackermann m n)`.
     Notes:

- To distinguish the two cases in `ackermann_mem`, add the following `display` command for the case when the input `(m n)` is in the current association list. It displays the string on screen.

```
(display ``memoization hit \n'')
```

- To add an entry to `al`, you will have to use `set!` to modify the global variable `al`. This has the side effect of modifying `al` so that it is visible to the next invocation of `ackermann_mem`.
- You will also need to use the sequencing construct in Racket. In particular, `(begin e1 e2)` evaluates `e1` (which usually has some side effect) and then evaluates `e2`; the value of `e2` becomes the value of `(begin e1 e2)`. For example,

```
(begin
   (display ``memoization hit \n'')
   (+ 1 2))
```

The example displays the message and returns 3.

2. (3 points). In this task, we write a function `construct_mem` that performs automatic memoization. It takes an input function with two arguments and returns a function that is the memoized version of the input function. That is `(construct_mem ackermann)` should return a function behaving the same as `ackermann_mem`. For simplicity, assume the input function takes exactly two parameters. As before, the returned function should display the message "memoization hit" when given an input that is already in the association list.

   Hint: the function returned by `construct_mem` should have its own association list, which cannot be defined as a global variable. Introduce a local variable instead. That is, the function `construct_mem` should be defined in the following way:

```
(define (construct_mem f)
   (let ((al '()))
      (lambda (m n) ... )))
```

   If you define `construct_mem` correctly, then the following sequence of code

```
(define ackermannm (construct_mem ackermann))
(ackermannm 2 1)
(ackermannm 1 1)
(ackermannm 2 1)
```

should display

```
5
3
memoization hit
5
```