CSSE304 Exam 2 Paper Review

Instructions

Please complete the following problems. If you have questions, come up and ask. I will send out an answer key after the review session is over.

1. **Problem 1:** What are the basic steps and extra pieces of your interpreter? Explain what each does. Steps:
   parse: breaks the incoming code into a new format that is more readable by the rest of our interpreter to make logic later on easier.
   syntax expand: turns more complicated procedures that our interpreter needs to handle into simpler more basic functions like lambda, if, and letrec.
   evaluate expression: this takes in the parse simple expressions and evaluates them using actual racket code.
   Other pieces:
   Environments: these are used to store variables just like the underlying environments that exist in racket. along with the local environments there is also a global environment that store basic procedures and can be modified.
   continuations: these store information about what is supposed to happen next after a specific CPS call.

2. **Problem 2:** What makes a function tail recursive or non-tail recursive? A function call is considered tail recursive if it the last thing being called in a function meaning it is not within any other functions and will be the value returned unmodified. Any other calls are non-tail recursive.

3. **Problem 3:** What is the goal of converting your interpreter to continuation passing style? The goal by doing this is to first of all allow the interpreter to handle call/cc. the other benefit is that CPS functions since they are all tail recursive run much faster making the interpreter work better.

4. **Problem 4:** How does call/cc work? call/cc works by taking in a lambda function that expects one argument. This argument is handled by the call/cc call which gives it the next continuation of the code. This next continuation is just what happens after the call/cc call. Then the body of the lambda is run and as it is being run if there is ever a call to the continuation (the input parameter) the function immediately stops running and returns to the point of the continuation. the call to the continuation can take one parameter which acts as the return value.

5. **Problem 5:** Walk through how this code runs. What does it output?

   ```
   (* 2 (call/cc
       (lambda (k)
           (+ 5 4 (k (- 10 4)) 4))))
   ```

   first it evaluates the 2 then goes into the call/cc function. the call/cc function gets a continuation that includes evaluating the 4 and the multiplication. inside of the call/cc function it evaluates the 4 then the 4 and then the k call. inside the k call it handles the subtract. The k call returns the the continuation passed into the call/cc function and evaluates the 4 and the multiplication. The multiplication gets the values 2 and 6. the 6 coming from return value of the call/cc function. the last 4 within the call/cc function is never run. the output is 12.

Imagine that we are in a lovely cafe in Paris. With us we have a genie who takes photographs of the things that are happening.

Here we are ordering our food. And in this one we are eating a delicious dessert. Now we are paying the bill.

Later, when we are remembering that wonderful night, we can ask the genie to rub a magic liquid on one of the photographs, and we will be back there in that cafe.
The waiter may be a bit older, we may be a lot heavier, but otherwise things are as they were before.

When we have finished our meal, we can spend the rest of the eveing wandering through the streets of Paris as we did before, or we can ask the genie to rub another photograph to take us somewhere else.

What if the Genie rubbed the wrong photograph and took us back to when we were paying the bill and leaving the restaurant? What a shame to have to pay for a meal that we never got to enjoy?

A Scheme interpreter is like a genie. It can remember where it is and go back there. It doesn't photograph everything, but only the things we tell it to photograph. It has a special "take a photograph" procedure named call-with-current-continuation. Applying call/cc produces a photograph (continuation) that we can rub (apply) whenever we want to escape to that point in the computation.