Interpreter

Lucas Larsson

Spring Term 2022

1 Introduction

Usually no intro is needed because the person reading this has done the same assignment, but i did an exception today since this assignment was one of the most complected and confusing this far, for me at least. Our assignment was to implement an `Interpreter`, for those like me how didn't know what is an interpreter. It is a program that is supposed to translate functions from a program language to another. An Example from one of my classmates is: think of at as a program that make is it possible for you to write a program in `Java` and run it in C, it could be for more efficiency or more readability or any reason, the reason don't really matter here, the important instance is that you write a program in syntax of a programming language, and run it on a different language platform.

Description

Once i got a basic idea about what the assignment is, I had a base to build on from a similar previous assignment, the derivative assignment, where there we needed to solve math deriving functions and here it is about Elixir functions. As usual not a lot of code will be shown here, but mostly an explanation of how to solve the assignment.

As per exercise documentation provided by the teacher, After i imported the skeleton program from the previous assignment I focused first on the two problem we needed to tackle, expression representation and the environment. Expression I Had already used in the Deriving assignment so it was not that difficult, and for that reason i will not be explaining more about it here. The environment on the other hand was a bit tricky to understand, but once i reread the part on it in the exercise documentation i understood it, it is basically more of the same as we did before, representing the different variables we get into the program with data structures that can manipulated later on in the program. And as mentioned in the documentation, the advantage of implementing a semi-Meta interpreter is that the data structures can be mapped directly to the same data structures used, because it is the exact same language i.e same data-structures.

Implementation

As soon as the idea and the planing phase was out of the way i got to writing functions, down below are some examples with explanation. I added some comments in the code to be easier to follow the explanation. Inspiration is also taken from this GitHub Repository.

```
# An environment
@type env :: [{atom, str}]

# Evaluation of a sequence
@spec eval(seq) :: {:ok, str} | :fail

# Creating a new empty environment
def new() do
    []
end

# Creates a new environment, this function is connected with the
# above listed function new()
def eval(seq) do
    eval_seq(seq, Env.new())
end

# This function evaluates an sequence for a environment and a program.
def eval_seq([{:match, ptr, exp} | seq], env) do
    case eval_expr(exp, env) do
    :error ->
        :error

    {:ok, str} ->
        env = eval_scope(ptr, env)

        case eval_match(ptr, str, env) do
        :fail ->
            :error

        {:ok, env} ->
            eval_seq(seq, env)
            end
        end
    end
```

This functions are from the file called `Eager`, the skeleton code provided was very helpful and helped guid me to the solution

Summary

This assignment was one of the most demanding so far, not sure because it is a high grade assignment or because it just get more complicated assignments as we develop in the course, but it was definitely a cool project to work on.

I feel it is a shame that not much was discussed in this report due to tight deadlines and exams in other courses.