Come on! Do math!

Praanto Samadder

January 31, 2024

1 The environment

In a previous assignment, we have used created an `environment`, a key-value map. For the sake of inefficiency, we shall trust José Valim and use the built-in implementation of the `Map` feature.

```
defmodule Environment do
    def new do
      Map.new()
    end

    def new(env) do
      Map.new(env)
    end
end
```

Listing 1: For loop that executes each term

And we're a bunch of lazy programmers so we will be using the `Map.get` function to lookup our elements in our environment.

2 `Expr` class

We use recursion on this task. We have five functions: `eval/2`, `add/2`, `sub/2`, `mul/2` and `divide/2`.

`eval/2` is called recursively until we reach a `{:num, val}` at which point arithmetic operations are done.

```
defmodule Expr do
    def eval({:num, n}, _) do
        {:num, n}
    end

    def eval({:var, x}, env) do
        Map.get(env, x)
    end

    def eval({:add, arg0, arg1}, env) do
        add(eval(arg0, env), eval(arg1, env))
    end

    def eval({:sub, arg0, arg1}, env) do
        sub(eval(arg0, env), eval(arg1, env))
    end

    ...
```

Listing 2: For loop that executes each term

There are 6 instances of `eval/2`: one for returning a `:num` tuple, one for looking up a variable in our environment and the other four for calculating arithmetic operations.

```
defmodule Expr do
    ...

    def add({:num, f1}, {:num, f2}) do
      {:num, f1 + f2}
    end


    def sub({:num, f1}, {:num, f2}) do
      {:num, f1 - f2}
    end

    def divide({:num, f1}, {:num, f2}) do
      simplify({:q, f1, f2})
    end

    ...
end
```

Listing 3: For loop that executes each term

We also have functions that will simplify our rational tuple. We can achieve this by first calculating the highest common factor of our numerator and our denominator and then dividing both sides of our fraction with that value.

3 Testing

We can express the equation $\frac{x}{y} + 4xy$ where $x = 5$ and $y = 3$ with

```
env = Environment.new([{:x, {:num, 5}}, {:y, {:num, 3}}])

expression = {:add, {:div, {:var, :x}, {:var, :y}}, {:mul, {:mul,
    {:var, :x}, {:var, :y}}, {:num, 4}}}
```

Listing 4: Testing our expression module

which gives us `{:q, 165, 3}` as a result.