Evaluation of an Expression

Ayda Ghalkhanbaz

Spring Term 2024

Introduction

This task involves implementation of a program that evaluates a mathematical expression containing variables and returns its result given an environment for the variables. This report covers the implementation details of the program and illustrates its functionality in a later section.

Method

The main function `eval/2` has already a predefined skeleton in the assignment. So, the first considerations included filling in the blanks and having base cases for each type, for example, evaluation of a number results in returning the number while evaluation of a variable includes the fetching of its value from the environment utilising the `Map` module. However, evaluating a quotient requires special handling, since the quotient should be reduced to the lowest common denominator, as well as not being divided by zero. As it can be seen in the code snippet below, the `eval/2` for the quotients employs if-else statements to handle these scenarios. The simplify/2 function reduces a fraction by dividing the numerator and the denominator by their greatest common divisor (gcd).

```
  def eval({:q, n, m}, _) do
      if m == 0 do
        {:error, "Division with 0"}
      else
        simplify(n,m)
      end end
####### Reduce & Simplify Functions #######
  def gcd(n, 0) do n end
  def gcd(0, m) do m end
  def gcd(n, m) do gcd(m, rem(n,m)) end
```

```
  def simplify(n, m) do
  factor = gcd(n,m)
  {:q, floor(n/factor), floor(m/factor)}
  end
```

After handling the base cases, the `eval/2` functions for operating on expressions were implemented. These functions utilise recursive calls to `eval/2` until a number, a value corresponding to the variable or a quotient is returned. Once the expressions are evaluated, the relevant operator is applied to perform the operation on the numbers. However, in case of division, the `eval/2` must control the potential "*division by zero*" error, which is done by having a `case-do` statement. The code below shows this implementation.

```
def eval({:div, e1, e2}, map) do
  case eval(e2, map) do
    {:num, 0} -> {:error, "Division by 0"}
    _ -> divz(eval(e1, map), eval(e2, map))
  end
end
```

The arithmetic functions `add/2`, `sub/2`, `mul/2` and `divz/2` perform the actual calculations on the passed numbers through the function calls. These functions manage different combinations of numbers and quotients and then send the result of the calculation to the `simplify/2` function for reduction in the case of quotients. The code overview in below, demonstrates the implementation of the `mul/2` function which handles various combinations between numbers and quotients.

```
def mul({:num, n}, {:num, m}) do {:num, n * m} end
def mul({:num, n}, {:q, a, b}) do
  {:q, w = (n*a), b}
  simplify(w, b)
end
def mul({:q, a, b}, {:num, n}) do
  {:q, w = (a*n), b}
  simplify(w, b)
end
def mul({:q, a, b}, {:q, c, d}) do
  {:q, n = (a*c), m = (b*d)}
  simplify(n, m)
end
```

Result

To confirm the functionality of the program, three different expressions have been tested. You'll find the result of the run tests below.

```
iex(2)> Evaluation.test()

The Environment: %{r: 5, w: 4, y: 6, x: 3, z: 2}

-------- Test1 ----------
The Expression: ((2*x) + r + 3/11) / (6)
The Result: 62/33

-------- Test2 ----------
The Expression: (3/4 - (1/4*y) + 7) / (10)
The Result: -31/40

-------- Test3 ----------
The Expression: ((4*z) + w) / (0)
The Result: Error, (Division by 0)
:ok
```
