Evaluating an expression

Sareh Jalalizad

Spring Term 2023

Introduction

For this assignment, we were supposed to implement a simple program that evaluates mathematical expressions that contain variables. The program should be able to take in a set of defined variables, referred to as an "environment," and use these variables to compute the result of the given expression.

Expressions

In this assignment, the representation of expressions is similar to that of the previous assignment, "Derivative", we create tuples for all expressions/variables instead of having a string. The literals we will use can be either integers, variables, or rational numbers. The program is limited to only four arithmetic operations: addition (:add), subtraction (:sub), multiplication (:mul), and integer division (:div).

```
@type literal() :: {:num, number() }
| {:var, atom()}
| {:q, number(), number()}

@type expr() :: {:add, expr(), expr()}
| {:mul, expr(), expr()}
| {:div, expr(), expr()}
| {:sub, expr(), expr()}
| literal()
```

Environment

Our task in evaluating an expression is to implement a function that takes both an expression and an environment as input and returns a literal as the result of the evaluation. The environment serves as a mapping from variable names to their respective values, we expect that all variables in

the expression have corresponding values. The environment should include two functions - one for creating a new environment with a given set of bindings and another for finding a binding given the name of a variable. The environment is structured as a list, with each binding represented as a tuple in the form of {{:var, var}, {:num, val}}, where var is the variable name and val is its corresponding value.

```
def env({:var, var}, []) do {:var, var} end
def env({:var, var}, [{{:var, var}, {:num, val}} | _]) do
    {:num, val} end
def env({:var, var}, [_|t]) do env({:var, var}, t) end
def env([{var, val}]) do [{{:var, var}, {:num, val}}] end
def env([{var, val} | t]) do [{{:var, var}, {:num, val}}
| env(t)] end
```

Evaluation

The next step is to create an eval function that utilizes the implemented environment. The implementation of the eval function is rather straightforward. There are some base cases to consider, such as returning the value of a number as-is, and returning the evaluated value of a variable that can be found in the environment. To improve readability, rational numbers should be simplified as much as possible and we can do it by using the **simplify** function so when we have a rational number then simplify function was called.

```
def eval({:num, num}, _) do {:num, num} end
def eval({:var, var}, env) do env({:var, var}, env) end
def eval({:q, num, denom}, _) do simplify({:q, num, denom}) end

def eval({:add, e1, e2}, env) do
 eval(add(eval(e1, env), eval(e2, env)), env) end
def eval({:sub, e1, e2}, env) do
 eval(sub(eval(e1, env), eval(e2, env)), env end

def simplify({:q, _, 0}) do :undefined end
def simplify({:q, 0, _}) do {:num, 0} end
def simplify({:q, num, denom}) do
  if rem(num, denom) == 0 do
    {:num, div(num,denom)}
  else
    gcd = Integer.gcd(num, denom)
    {:q, div(num, gcd), div(denom,gcd)} end
end
```

In order to work with fractions, it was necessary to implement the rules of arithmetic for various operations involving them. As the fundamental operations are similar to others, they will not be mentioned. These functions were used for operations multiplication and division:

```
def mul({:num, 0}, _) do {:num, 0} end
def mul(_, {:num, 0}) do {:num, 0} end
def mul({:num, n1}, {:num, n2}) do {:num, n1 * n2} end

def mul({:q, num1, denom1}, {:q, num2, denom2}) do
    simplify({:q, (num1 * num2), (denom1 * denom2)}) end

def mul({:num, n}, {:q, num, denom}) do
    simplify({:q, (n * num), denom}) end

def mul({:q, num, denom}, {:num, n}) do
    simplify({:q, (n * num), denom}) end


def division({:num, 0}, _) do {:num, 0} end
def division(_, {:num, 0}) do :undifined end

def division({:num, n1}, {:num, n2}) do
    simplify{:q, n1, n2} end

def division({:q, num1, denom1}, {:q, num2, denom2}) do
    simplify({:q, (num1 * denom2), (num2 * denom1)}) end

def division({:num, n}, {:q, num, denom}) do
    simplify({:q, (n * denom), num}) end

def division({:q, num, denom}, {:num, n}) do
    simplify({:q, num, (n * denom)}) end
```

Conslusion

Overall, this assignment was a relatively straightforward task that demonstrated the process of evaluating an expression with a given environment. Initially, the assignment might have seemed confusing and challenging, but after conducting further research and gaining a better understanding of the requirements, it became easier to complete.