# An environment

Emil Göransson

Spring Term 2023

## Introduction

This report will cover the implementation of a program that given a mathematical expression containing variables, will evaluate and calculate the expression by cross-checking a database containing variable-value pairs.

## Explanation

In order to calculate the expression we need two things. First of all we need an expression written in the way previously known to us. One example of this would be the expression

$$2x + 3 + \frac{1}{2} \tag{1}$$

. In order to calculate it we write it in the following way:

```
expr = {:add, {:add, {:mul, {:num, 2}, {:var, :x}}, {:num, 3}}, {:q, 1,2}}
```

The way we write the expression comes from the way we defining the inputs.

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

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

The issue we have here is that given an variable defined as such:

```
{:var, :x}
```

We need a way to cross-check this variable in a given key-value database. For this we use the built in data-structure "Map". Using the following syntax we are able to save the variable pair x-5 and y-2 in the "map" env.

```
env =  %{x: 5, y: 2}
```

Now using our expression eval/2 to evaluate the expression and simplify to simplify and style the output we are able to calculate the actual value of the entered mathematical expression by calling the functions like this:

```
simplify(eval(expression, env))
```

eval/2 works in such a way that using pattern-matching it can differentiate between subtraction, addition, variables,multiplication, and quotients. Every method pretty much has the same structure, with only the mathematical rules differentiating them. Because of this I will focus on add, since if you know add, you should also understand the other 4 operators respective methods since they are all very similar. Because we need to be able to deal with rational numbers, we make sure to always return the values in the form: {:q, x, y} where:

$$x, y \in Z \tag{2}$$

so that simplify can simplify the answer.

### Some important "base"-cases

The two base cases that every call will use at least once are the following two:

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

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

In the case of a :num (number) we return the value of the number n.

In the case of a :var (variable) we check the environment (map) for the entered value and return the value part of the key-value pair if found.

Becase we need to be able to handle quotients as stated in the assignment the following bit of code is used to get the quotient in the simplest form possible

```
  def simplify({:q, e1, e2}) do
  gcd = Integer.gcd(e1, e2)

  if(gcd == 1) do
    pprint({:q, e1, e2})
  else
    pprint({:q, e1 / gcd, e2 / gcd})
  end
end
```

### Addition and Quotients

Inside eval we use pattern matching. In the case of the input being of the :add type (our way of defining addition) the following snippet of code is ran.

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

Here we use recursion to find the "actual value" of e1 and e2 before using it in the add/2 function.

#### add/2

add/2 has 4 "cases. Though pattern matching the correct one is selected. The reason 4 cases in needed is because we need to handle the case we get either one or two quotients (:q). The code is the following:

```
def add({:q, e1, e2}, {:q, e3, e4}) do
  {:q, e1 * e4 + e3 * e2, e2 * e4}
end

def add({:q, e1, e2}, e3) do
  {:q, e1 + e3 * e2, e2}
end

def add(e1, {:q, e2, e3}) do
  {:q, e1 * e3 + e2, e3}
end

def add(e1, e2) do
  {:q, e1 + e2, 1}
end
```

The code above is pretty simple, it uses the rules of addition and because quotients are a thing, every case will always return a quotient in some sort of way.

## The code in action

Some equations and its respective results using the map %{x: 1, y: 2}

### Equations and its inputs

1. $$2x + 3 + \frac{1}{2} \tag{3}$$

```
{:add, {:add, {:mul, {:num, 2}, {:var, :x}}, {:num, 3}}, {:q, 1, 2}}
```

2. $$2x - \frac{3}{4} + \frac{1}{2} \tag{4}$$

```
{:sub, {:add, {:mul, {:num, 2}, {:var, :x}}, {:q, 3, 4}}, {:q, 1, 2}}
```

3. $$\frac{2y}{\frac{3}{4} + \frac{1}{2}} \tag{5}$$

```
{:div, {:mul, {:num, 2}, {:var, :y}}, {:add, {:q, 3, 4}, {:q, 1, 2}}}
```

### Outputs

1. 13/4
2. 9/4
3. 16/5