Evaluating an expression

Programming II - Elixir Version

Johan Montelius

Spring Term 2023

Introduction

In this assignment you will evaluate a mathematical expression containing variables.

Expressions

Arithmetic expressions are represented as tuples {`op, arg1, arg2`} and we can for the time being limit ourselves to the operators `:add`, `:sub`, `:mul` and integer division `:div`. Expressions are thus:

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

The literals that we will use are either integers, variables or rational numbers. To make it explicit we choose to represent integers as {`:num, n`} and variables as {`:var, a`}. Rational numbers are represented as {`:q, n, m`}.

This gives us everything we need to represent a limited sets of expressions. The expression $2x + 3 + 1/2$ could for example be represented by the Elixir structure:

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

As you see it it is not a syntax we would like to use when we write expressions by hand but it has its advantages when it comes to handle the expressions using Elixir clauses.

Evaluation

You task is to implement a function `eval/2`, that takes an expression and an environment and evaluate the expression to a literal. The environment is a mapping from variable names to values and we expect to have values for all variables in the expression.

The environment should provide two functions, one to create a new environment with a given set of bindings and one function that finds a binding given a variable name.

Once you have an environment working then the function `eval/2` should be done in fifteen minutes, this skeleton might give you a flying start:

```
def eval({:num, ..}, ...) do ... end
def eval({:var, ..}, ...) do ... end
def eval({:add, ..., ...}, ...) do
  add(..., ...)
end
  :
  :
```

If you follow this skeleton you only have to implement the function `add/2`, `sub/2` etc. It seams like a trivial task and this is why we threw in rational numbers and integer division. Dividing 5 by 2 is thus not 2.5 but 2/5 or as we would represent it `{:q, 2, 5}`. So when you implement the arithmetic operations you have to take into account that one of the operands might be a rational number.

In order to make tings more readable we of course want to reduce the rational numbers as much as possible. If you evaluate $2 \times 3/4$ the answer should not be 6/4 but 3/2.

Implement the function `eval/2` and show by some examples that it works.