ID1019: Derivative

Alexander Lundqvist

Spring Term 2023

Introduction

In this task we are supposed to create an Elixir program that can take the derivative of some common mathematical expressions such as addition, multiplication, division, square root, exponential, logarithms and trigonometric according to the following rules.

- $\frac{d}{dx}x \equiv 1$
- $\frac{d}{dx}c \equiv 0$ for any literal different from x
- $\frac{d}{dx}(f + g) \equiv \frac{d}{dx}f + \frac{d}{dx}g$
- $\frac{d}{dx}(f * g) \equiv \frac{d}{dx}f * g + \frac{d}{dx}g * f$

Method

To accomplish this task I followed the video instructions and instruction document provided. During the process I also utilized the Elixir documentation to understand what I was actually doing. The initial work was to set up the basic skeleton of the program and the initial derivation methods. The basic structure of the program is the the derivation method with corresponding simplification and print method. Here we have an example how the methods work for each type of derivation.

```
def derive({:add, expression1, expression2}, x) do
    {:add, derive(expression1, x), derive(expression2, x)}
end
def simplify({:add, expression1, expression2}) do
  simplify_add(simplify(expression1), simplify(expression2))
end
def simplify_add(expression1, expression2) do
    {:add, expression1, expression2}
end
```

```
    def pprint({:add, expression1, expression2}) do
        "(#{pprint(expression1)} + #{pprint(expression2)})"
    end
```

Lastly I implemented several test functions (test1, test2, test3...) to test and verify the behaviour of the program. The code and the test results are mentioned in the result section.

Result

```
    @doc """
    Find a derivative of given expression.
    """
    def derive(expression) do

    end
```

The result from running the tests are listed below:

```
Interactive Elixir (1.14.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> c("Derivative.ex")
[Derivative]
iex(2)> Derivative.test1()
-------- Testing addition --------
Expression: (3*(x)^(2) + 4*x)
Derivative of expression: ((0*(x)^(2) + 3*2*(x)^(1)*1) + (0*x + 4*1))
Simplified: (3*2*x + 4)

:ok
iex(3)> Derivative.test2()
----- Testing multiplication -----
Expression: 3*x*4*x
Derivative of expression: ((0*x + 3*1)*4*x + 3*x*(0*x + 4*1))
Simplified: (3*4*x + 3*x*4)

:ok
iex(4)> Derivative.test3()
--------- Testing division -------
Expression: (3/x)
Derivative of expression: ((0*x + 3*1*-1)/(x)^(2))
Simplified: (-3/(x)^(2))

:ok
iex(5)> Derivative.test4()
```

```
------- Testing exponential ------
Expression: (x)^(3)
Derivative of expression: 3*(x)^(2)*1
Simplified: 3*(x)^(2)

:ok
iex(6)> Derivative.test5()
------- Testing square root ------
Expression: sqrt(5*x)
Derivative of expression: ((0*x + 5*1)/2*sqrt(5*x))
Simplified: (5/2*sqrt(5*x))

:ok
iex(7)> Derivative.test6()
----------- Testing ln -----------
Expression: ln((x)^(10))
Derivative of expression: (10*(x)^(9)*1/(x)^(10))
Simplified: (10*(x)^(9)/(x)^(10))

iex(8)> Derivative.test7()
----------- Testing sin ----------
Expression: sin(x*3)
Derivative of expression: (1*3 + x*0)*cos(x*3)
Simplified: 3*cos(x*3)

:ok
```

The entire code is available at this github repository.

Discussion

The results prove that the program behaves in the intended manner an the tests can be verified through either Wolfram Alpha or Symbolab. The output is unfortunately not formatted in a way that I would've wanted and I would've liked to implement an actual parser that could handle user input. When researching this matter however I realised that the amount of work required to construct an actual parser was too much for what was actually required for this task. The task itself wasn't particularly difficult as the instructions provided a clear way how to implement the program.