Derivative

Sareh Jalalizad

Spring Term 2023

Introduction

For this assignment, we were supposed to create a simple program that, with the aid of pattern matching, could take the derivative of a mathematical function and display output as a simplified version of that derivative.

Expressions

First, we had to analyze how the functions should be structured. We create tuples for all expressions/variables instead of having a string. As illustrated below, the first item is an atom to identify the type of expression it is:

```
(x / (x + 2)) =>
     {:frac, {:var, :x}, {:add, {:var, :x},{:num, 2}}}
```

We also decided to work with a total of 2 literal types (number, variable) and 7 expressions (addition, multiplication, exponent, fraction, square root, ln, and sin).

Derivatives

The next step is to create a derivative function which would derive the given expression. There are several types of mathematical expressions that the program should be able to manage, so when we know thoroughly how the derivation rules work, the implementation is rather straightforward. The different types of functions will be covered in this section. Because the some fundamental operations were demonstrated in the lecture and some of them are similar to others, they won't be mentioned.

By dividing the derivative of the expression by the expression, the derivative of the **natural log** function is calculated.

```
def deriv({:ln, e}, v) do
   {:frac, deriv(e,v), e}
end
```

The denominator times the derivative of the numerator minus the numerator times the derivative of the denominator, all divided by the square of the denominator, makes up the derivative of a **division**. This is a general answer, but there's also the option of writing fractions as exponents, as in $(x^{-}1)$ for $1/x$.

```
def deriv({:frac, e1, e2}, v) do
   {:frac, {:add, {:mul, e2, deriv(e1, v)},
   {:mul, {:num, -1}, {:mul, e1, deriv(e2, v)}}},
   {:exp, e2, {:num, 2}}}
end
```

The derivative of the **square root** is calculated by the division of the derivative of the expression by twice the original expression.

```
def deriv({:sqrt, e}, v) do
   {:frac, deriv(e, v),
   {:mul, {:num, 2}, {:sqrt, e}}}
end
```

By multiplying the derivation of the expression inside the sin function by the cosine of the function, the derivative of the **sin** function is calculated.

```
def deriv({:sin, e}, v) do
   {:mul, deriv(e,v), {:cos, e} }
end
```

Simplification

The problem with the derivative function is that the output is displayed as a tuple and that many unnecessary literals were printed to the console, which makes it difficult for the user to understand and read the output easily. Depending on the expression, it is frequently possible to simplify it by writing different functions to represent what the function should return in specific cases, such as by eliminating zeros or variables that have been multiplied by one. The code for different cases is shown below:

```
def simplify_frac({:num, 0}, _) do {:num, 0} end
def simplify_frac(e1, {:num, 1}) do e1 end
def simplify_frac(_, {:num, 0}) do :error end
def simplify_frac({:num, n1}, {:num, n2}) do {:num, n1/n2} end
def simplify_frac({:var, v}, {:var, v}) do {:num, 1} end
def simplify_frac(e1, e2) do {:frac, e1, e2} end

def simplify_sin({:num, 0}) do {:num, 0} end
def simplify_sin({:num, n}) do {:num, :math.sin(n)} end
def simplify_sin(e1) do {:sin, e1} end
def simplify_cos({:num, 0}) do {:num, 1} end
def simplify_cos({:num, n}) do {:num, :math.cos(n)} end
def simplify_cos(e1) do {:cos, e1} end

def simplify_sqrt({:num, 0}) do {:num, 0} end
def simplify_sqrt({:num, 1}) do {:num, 1} end
def simplify_sqrt({:num, n}) do {:num, :math.sqrt(n)} end
def simplify_sqrt(e1) do {:sqrt, e1} end
```

Then we create the function **pprint** in order to display the expression in a more readable manner. The function is called recursively in the string, and printing the different variables according to the operation continuously. Below is an illustration of printing the functions:

```
def pprint({:frac, e1, e2}) do "(#{pprint(e1)} / #{pprint(e2)})" end
def pprint({:ln, e}) do "ln(#{pprint(e)})" end
def pprint({:sin, e}) do "sin(#{pprint(e)})" end
```

Conslusion

Overall, this assignment was interesting and which shows how to use recursion and pattern matching. Tuples were at first confusing and quite difficult to understand, however, this task provided a great illustration of how to use them.