Derivatives

Dhruba Das

Spring Term 2024

Introduction: Problems to solve

There are in total 3 problems I had to solve in this assignment: Implementing the types of data, implementing the rules of the derivatives, and simplification of the obtained expressions-

Data types

I had to define the types for numbers, atoms and different expressions, as shown below:

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

  @type expr() :: {:add, expr(), expr()}
  | {:mul, expr(), expr()}
  | {:exp, expr(), literal()}
  | {:ln, expr()}
  | {:sin, expr()}
  | {:cos, expr()}
  | literal()
```

Rules of derivation

This wasn't actually the trickiest part of the task, with only adding the four rules for derivation, along with new methods with minor changes which use recursion to derive/differentiate the given expression; note that the reciprocal and square roots derivation is done using the : *exp* data type. Below is the code:

```
def deriv({:num, _}, _) do {:num, 0} end
  def deriv({:var, v}, v) do {:num, 1} end
  def deriv({:var, _}, _) do {:num, 0} end
  def deriv({:add, e1, e2}, v) do
    {:add, deriv(e1,v), deriv(e2,v)}
  end
  def deriv({:mul, e1, e2}, v) do
    {:add,
     {:mul, deriv(e1, v), e2},
      {:mul, e1, deriv(e2, v)}}
  end
  def deriv({:exp, e, {:num, n}}, v) do
    {:mul,
     deriv(e,v),
      {:mul, {:num, n}, {:exp, e, {:num, (n-1)}}}}
  end
  def deriv({:ln, e}, v) do
    {:mul,
     deriv(e, v), {:exp, e, {:num, -1}}}
  end
  def deriv({:sin, e}, v) do
    {:mul, deriv(e, v), {:cos, e}}
  end
```

Simplification

This was the toughest part of the task for me, although not actually tough since I'd gone through all 6 instruction videos before attempting the assignment. Shown below is the code for simplifying different expressions, including those for addition, multiplication, natural log (*ln*), and *sin* and *cos* (since there is a lot of code, I'll only be including the code for the crucial bits, since other lines of code are there for pattern-matching)

```
def simplify({:num, n}) do {:num, n} end
def simplify({:var, v}) do {:var, v} end
...
def simplify({:add, e1, e2}) do
  simplify_add(simplify(e1), simplify(e2))
end
def simplify({:mul, e1, e2}) do
  simplify_mul(simplify(e1), simplify(e2))
end
def simplify({:exp, e1, e2}) do
  simplify_exp(simplify(e1), simplify(e2))
end
...
```

```
def simplify_mul({:num, 0}, _) do {:num, 0} end
...
def simplify_mul({:num, 1}, e2) do e2 end
...
def simplify_mul({:num, n1}, {:num, n2}) do {:num, n1*n2} end
...
def simplify_mul({:num, n1}, {:mul, {:num, n2}, e2}) do
  {:mul, {:num, n1*n2}, e2}
end
...
def simplify_mul({:mul, {:num, n2}, e2}, {:num, n1}) do
  {:mul, {:num, n1*n2}, e2}
end
def simplify_mul({:mul, e1, {:num, n2}}, {:num, n1}) do
  {:mul, {:num, n1*n2}, e1}
end

def simplify_mul(e1, e2) do {:mul, e1, e2} end
...
def simplify_exp(_, {:num, 0}) do {:num, 1} end
def simplify_exp(e1, {:num, 1}) do e1 end
def simplify_exp({:num, 0}, _) do {:num, 0} end
def simplify_exp({:num, 1}, _) do {:num, 1} end
def simplify_exp({:num, n1}, {:num, n2}) do {:num, :math.pow(n1,n2)} end
def simplify_exp(e1, e2) do {:exp, e1, e2} end
```

Print functions

Shown below are also the print functions which I implemented:

```
def pprint({:num, n}) do "#{n}" end
def pprint({:var, v}) do "#{v}" end
def pprint({:add, e1, e2}) do "(#{pprint(e1)} + #{pprint(e2)})" end
def pprint({:mul, e1, e2}) do "#{pprint(e1)} * #{pprint(e2)}" end
def pprint({:exp, e1, e2}) do "#{pprint(e1)}^(#{pprint(e2)})" end
def pprint({:ln, e1}) do "ln(#{pprint(e1)})" end
def pprint({:sin, e1}) do "sin(#{pprint(e1)})" end
def pprint({:cos, e1}) do "cos(#{pprint(e1)})" end
```

An example

Shown below is an example which uses the rules and simplification implemented:

```
def test1() do
  b = {:mul, {:num, 1}, {:exp, {:sin, {:mul, {:var, :x}, {:num, 2}}}, {:num, -1}}}
  c = deriv(b, :x)
  d = simplify(c)
  IO.write("expression: #{pprint(b)} \n")
  IO.write("derivative: #{pprint(c)} \n")
  IO.write("simplified: #{pprint(d)} \n")
  d
 end
```

Which gives us the output:

```
expression: 1 * sin(x * 2)^(-1)

derivative:
(0 * sin(x * 2)^(-1) + 1 * (1 * 2 + x * 0) * cos(x * 2) * -1 * sin(x * 2)^(-2))

simplified: 2 * cos(x * 2) * -1 * sin(x * 2)^(-2)
```