PFL Project

João Pereira, Nuno Pereira

October 22, 2022

Abstract

TODO

Contents

1 Introduction **3**

2 Internal Representation **3**

A Appendix **4**

A.1 Figures 4

1 Introduction

The goal of this project was to implement polynomials and common operations performed on them, such as summation or derivation, using the Haskell programming language.

2 Internal Representation

For the internal representation of the polynomial data structure, we implemented the following:

```
import Data.Map

data Natural = One | Suc Natural deriving (Eq, Ord)

type Variable = Char

type Exponent = Natural

type Coefficient = Double

data Monomial = Monomial Coefficient (Map Variable Exponent)
    deriving (Eq)

newtype Polynomial = Polynomial [Monomial] deriving (Eq)
```

This allows us to represent *Polynionals* and *Monomials* in a way that naturally represents what they are, while also ensuring that the operations performed on them are efficient:

- doing work on a *Polynomial* is (almost) the same as doing the same work to each of its *Monomials*;
- working with the variables and degrees of each *Monomial* is not only simplified but also more efficient because of the nature of the underlying *Map* data structure (for example, normalizing a polynomial is done in $O(k'km*log(\frac{n+1}{m+1}+k'), m \leq n$ time instead of $O(k^2)$):
  - $O(m * log(\frac{n+1}{m+1}), m \leq n$ for aggregating any 2 *Monomials*, where n and m are the sizes of the *Monomials*' "exponent map";
  - $O(k-1) = O(k)$, where k is the number of *Monomials* in the original *Polynomial*;
  - $O(k' * log(k'))$ for sorting the aggregated *Monomials*, where *k'* is the number of *Monomials* in the *Polymial* that resulted from the previous step;

A Appendix

A.1 Figures

TODO