DM 509 Programming Languages

Fall 2009 Project (Part 2)

Department of Mathematics and Computer Science
University of Southern Denmark

December 7, 2009

Introduction

The purpose of the project for DM509 is to try in practice the use of logic and functional programming for small but non-trivial examples. The project consists of two parts. The first deals with logic programming and the second part with functional programming.

Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules

This second part of the project is a part of the final exam. Both parts of the project have to be passed to pass the overall project and get access to the final written exam.

Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Each part is passed when its deliverables are rated at 50%. Any percentage above 50% can be transferred between the individual parts, i.e., with 40% in the first part and 80% in the second part, by transferring 10% from the second part to the first part, both parts are passed.

Deliverables

There is one deliverable for this second part of the project:

- A short project report (2-4 pages) containing a description of your approach, a short description of the main parts of your source code, and a short documentation of the testing you have performed. You have to include an electronic copy of your source code.

The deliverable has to be delivered using Blackboard's Assignment Hand-In functionality. Delivering by e-mail or to the teacher is only considered acceptable in case Blackboard cannot be used.

Deadline

Deliverable: December 21, 12:00

The Problem

Your task in this part of the project is to work with a data structure for representing polynomial fractions, i.e., expressions built from variables, constants, addition, subtraction, multiplication, and division.

The data structure is defined in the following way:

```
data Operator = Add | Sub | Mul | Div
data Poly = C Float | V String | Op Poly Operator Poly
```

Using this data structure, we can for example represent the polynomial $4 - x + 3x^2$ by the following expression:

```
Op (Op (C 4) Sub (V "x")) Add (Op (C 3) Mul (Op (V "x") Mul (V "x")))
```

There is a template available from the course home page that defines this data structure and binds this expression to the variable `testpoly`.

This template also contains an incomplete definition of a function for evaluating variable-free polynomial fractions:

```
eval :: Poly -> Float
eval (Op p1 o p2) = interpret o val1 val2 where
    val1 = eval p1
    val2 = eval p2
```

The Tasks

Implement the following operations on polynomial fractions by implementing the following functions (and any auxiliary functions you might consider needed):

1. Implement the function `interpret` used in `eval` above to complete the definition of our evaluation function. Use the form given in the template.

2. Define a function `derive :: String -> Poly -> Poly` which computes the (symbolic) derivation of a polynomial fraction with respect to a given variable identified by its name. For example, the expression `derive "x" testpoly` should return a polynomial which corresponds to `-1 + 6x`.

3. Define a function `simplify :: Poly -> Poly` to simplify polynomial fractions. You can use rules like $0 + x = x$ and $0 * x = 0$ as well as the function `eval`. For example, `simplify (derive "x" testpoly)` should return a result at least as simple as $-1 + 3 * (x + x)$. And `simplify (Op (V "x") Div (C 1.0))` should evaluate to `V "x"`.

4. Define a data type `Substitution` that maps some variables to constant values. Then define a function of the type `instantiate :: Substitution -> Poly -> Poly` that takes a polynomial fraction and instantiates all variables mentioned in the substitution by the corresponding constant. After instantiating the variables, the resulting expression should be simplified. For example, if you call `instantiate` on `testpoly` with a substitution that maps x to 1.0, the result should be `C 6.0`.

Hint: By removing `deriving Show` behind the definitions of `Operator` and `Poly` and uncommenting the `show` declarations at the bottom of the template, you can view the polynomials in a more human-readable format.