Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Lenguajes de Programación
Enero-Marzo 2019

Proyecto 1: Implementación de Calculadora Simbólica de Antiderivadas Tigronométricas

1. Introducción

Se requiere que usted implemente usando lenguaje de programación Haskell, una calculadora simbólica de integrales para los estudiantes del curso de Matemáticas 2. Una calculadora simbólica es una aplicación que responde al usuario una serie de transformaciones algebraicas en un determinado orden, por ejemplo, los pasos algebraicos que permiten calcular la fórmula de una antiderivada.

Existen en la actualidad calculadoras simbólicas para el cálculo diferencial bastantes sofisticados, como es el caso de Mathematica [1], Maple [2], Wolfram Alpha [3], Symbolab [4]. En nuestro caso implementaremos una calculadora simbólica bastante sencilla, que permita calcular las antiderivadas de algunos polinomios trigonométricos de senos y cosenos.

2. Definición del lenguaje de polinomios trigonométricos

Sea un conjunto de variables Var que constan de todas las letras del abecedario, definamos los términos válidos de nuestro lenguaje:

- toda fracción $\frac{p}{q}$ con $p, q \in \mathbb{Z}$ es un término válido.
- Si $t \in Var$ entonces t es un término válido.
- Si t es un término válido, entonces $sen(t)$ y $cos(t)$ son términos válidos.
- Si t es un término válido y $x \in Var$, entonces $\int tdx$ es un término válido.
- Si t es un término válido y $n \in \mathbb{N}$, entonces t^n es un término válido.
- Si $t1$ y $t2$ son términos válidos, entonces $t1 + t2$, $t1 * t2$ son términos válidos.

La precedencia de los operadores es la misma que se usa para los polinomios en el cálculo.

3. Detalles de la implementación

3.1. Operadores

En un módulo en Haskell llamado `Term.hs`, usted debe implementar el conjunto de los términos válidos usando tipos algebraicos recursivos. Por ejemplo usted puede implementar al tipo `Term` como

```
data Term = Const Int Int| Var String | Fun String Term | Integ Term Term |
            Sum Term Term | Mult Term Term | Exp Term Int
```

Donde Const es el constructor de las fracciones con su numerador y denominador, Var el de las variables con su nombre, Fun el de las funciones seno o coseno con su argumento, Integ el de la integral con su expresión y variable, Sum el de la suma, Mult el de la multiplicacion y Exp el de la exponenciación.

Con este tipo de dato, debe instanciar la clase Num de Haskell, sobrescribiendo los operadores (+), (*) y fromInteger de tal forma que t1 + t2 se corresponda a Sum t1 t2, t1 * t2 a Mult t1 t2 y un número entero i dentro de una expresión que involucre Terms, se corresponda a Const i 1.

La operación * sólo debe simplificar el término cuando se encuentren enteros entre factores. Por ejemplo si x es una variable, entonces las expresiones 2*3*x, 2*(3*x) y 2*((3*x)*2) deben resultar en $6x$, $6x$ y $12x$ respectivamente.

Igualmente usted debe instanciar la clase Show, implementando la función show para los objetos de tipo Term. La impresión de los Term debe ser en el formato math mode de latex, como se explica a continuación:

- Una fracción cuyo numerador y denominador son p y 1 respectivamente se imprime como p
- Una fracción cuyo numerador y denominador son p y q respectivamente, con $q \neq 1$ se imprime como \frac{p}{q}
- Una variable (Var x) se imprime como el String x
- Una función (Fun f t) se imprime como f(<show t>)
- Una integral (Integ t1 x) se imprime como \int <show t> d<show x>
- Una suma se debe imprimir usando el operador + de forma infija
- Una multiplicación se debe imprimir de forma infija usando un espacio en blanco sin ningún operador para separa los argumentos
- Una exponenciación (Exp t1 n) se debe imprimir como <show t1>^{n} salvo cuando t1 es la función seno o coseno, donde la impresión es de la forma sen^{n}(<show argumento>)

La función show debe imprimir el término ahorrando paréntesis según las reglas de precedencia de los operadores del cálculo clásico.

3.2. Variables

Con la intención de poder ingresar fácilmente términos de tipo Term en la consola de ghci, usted definirá, en el módulo Term, por cada letra σ del alfabeto, de la t en adelante, una función constante de nombre σ que devuelve el objeto de tipo Term, que representa a la variable de letra σ. Por ejemplo:

```
t :: Term
t = Var "t"

u :: Term
u = Var "u"

v :: Term
v = Var "v"

w :: Term
w = Var "w"
    :
    :
    :
```

Igualmente usted debe agregar las definiciones de las funciones seno y coseno de la siguiente forma:

```
sen::(Term) -> Term
sen (t1) = Fun "sen" t1

cosen::(Term) -> Term
cosen (t1) = Fun "cos" t1
```

Con estas definiciones podremos escribir en la consola de ghci expresiones como `2*x*y + sen(y)^2+4*cosen(x^3)`, de manera que esta última expresión es interpretada por el interprete, como el término que representa a $2xy+sen(y)sen(y)+4cos(xxx)$ en el modelo concreto de su implementación (hecha con tipos algebraicos recursivos).

3.3. Álgebra

En un módulo llamado Algebra.hs debe usted implementar como mínimo las funciones siguientes sobre objetos de tipo `Term`

3.3.1. Simplificación

Usted debe implementar en el módulo Algebra, una función llamada `simplify` tal que dado un Término, hace una búsqueda por todos sus subtérminos y sustituye algunos subtérminos por otros que llamaremos simplificación. La regla de escogencia para hacer simplificación, se muestra en la siguiente lista, donde el lado izquierdo de la flecha indica el esquema de subtérmino que se debe cambiar y el lado derecho de la flecha indica el esquema de término que se debe colocar en su lugar.

- $t^0 \rightarrow 1$
- Si a es una fracción entonces $a^n \rightarrow resultado\ de\ a^n$
- Si a es una fracción entonces $a \rightarrow eliminar\ el\ maximo\ comun\ divisor\ entre\ el\ numerador\ y\ denominador$
- Si a y b son fracciones entonces $a + b \rightarrow fraccion\ resultante\ de\ sumar\ a\ y\ b$
- $0 + t \rightarrow t$
- $t + 0 \rightarrow t$
- $1 * t \rightarrow t$
- $t * 1 \rightarrow t$

La función `simplify` debe ser implementada de forma que realice todas las simplificaciones posibles, de modo que si una simplificación genera otro termino simplificable, este nuevo subtérmino debe ser simplificado también, es decir que con una sola llamada de `simplify`, se simplifican los términos simplificables y los términos simplificables resultantes de las simplificaciones anteriores.

3.3.2. Distributividad

Usted debe implementar en el módulo Algebra, una función llamada `distrib` tal que dado un Término, hace una búsqueda por todos sus subtérminos y sustituye los subtérminos de la forma $t1(t2+t3)$ y $(t2+t3)t1$ por $t1t2 + t1t3$ y $t2t1 + t3t1$ respectivamente.

La función `distrib` no debe realizar simplificaciones, salvo la de sustituir el resultado de la multiplicación de fracciones que aparezcan como factores, al aplicar la propiedad distributiva. Por ejemplo al encontrar el subtérmino $(y4 + 3z)2x$, éste debe ser sustituido por $8yx + 6zx$

La función `distrib` debe ser implementada de forma que realice todas las propiedades distributivas posibles, de modo que si una propiedad distributiva genera otro subtérmino distribuible, este nuevo subtérmino debe ser sustituido también con la misma llamada de `distrib`.

3.3.3. Agrupar Átomos

Definiendo que un átomo es cualquier término que sea una fracción, variable, función, integral o exponenciación de los anteriores, entonces se define que un término es un monomio, cuando es un átomo o multiplicación de átomos.

En el módulo Algebra, usted debe implementar una función llamada `groupAtoms`, que dado un Término, hace una búsqueda en todos sus subtérminos y cuando encuentra un monomio ordena y agrupa los átomos que tengan la misma base sumando sus exponentes o realizando la multiplicación, si los átomos son fracciones. Por ejemplo, cuando `groupAtoms` encuentra el monomio $2xy4sen^2(2x)xsen(2x)$, la función transforma éste en $8sen^3(2x)x^2y$. El orden en que se ordenan los átomos luego de aplicar `groupAtoms`, depende de la siguiente relación de equivalencia y de orden entre términos:

- Si $t1$ y $t2$ no son fracciones, entonces $t1 = t2$ si y sólo si $t1^n \equiv t2^m$ para todo $n, m \geq 1$
- si $t1$ y $t2$ son fracciones, entonces $t1 \equiv t2$

Es decir hay dos tipos de términos equivalentes, los que son equivalentes porque son fracciones y los que tienen la misma base. La función `groupAtoms` debe encontrar los términos equivalentes dentro de un monomio y efectuar la multiplicación entre los que son fracciones y sumar los exponentes entre los que tenga la misma base.

La relación de orden $<$ para decidir el orden en que los átomos de un monomio deben aparecer luego de ejecutar `groupAtoms`, se define de la siguiente manera:

- Una fracción es menor que cualquier término distinto de fracción
- Una integral es mayor que cualquier término distinto de integral
- Todo átomo distinto de integral es menor a un término que no sea átomo
- Un término suma es menor que un término multiplicación y exponenciación
- Un término multiplicación es menor que un término exponenciación
- Una variable es menor a otra si su nombre es alfabéticamente menor al nombre de la otra variable
- Una función es menor a otra si su nombre es menor alfabéticamente que el nombre de la otra función, en caso de que los nombres de las funciones sean iguales, se decide cual de las dos funciones es menor, dependiendo de cual de los dos argumentos de las funciones es menor
- Una integral es menor que otra, si su integrando es menor que el integrando de la otra integral
- Entre una variable y una función se decide cual de las dos es menor, dependiendo de cual de los nombres de la variable y la función es menor alfabéticamente
- Para cualquier $n, m \geq 1$ se tiene que $t1^n < t2^m$ si y solo si $t1 < t2$
- Entre dos términos $t1 + t2$ y $t3 + t4$ se decide cual de los dos es menor dependiendo de cual entre $t1$ y $t3$ es menor, en caso de ser iguales se pasa a comparar $t2$ y $t4$
- Entre dos términos $t1 * t2$ y $t3 * t4$ se decide cual de los dos es menor dependiendo de cual entre $t1$ y $t3$ es menor, en caso de ser iguales se pasa a comparar $t2$ y $t4$

3.3.4. Potenciación

En el módulo Algebra debe implementar una función llamada `pow`, que dado un Término, recorre todos sus subtérminos y cuando consigue una exponenciación, reemplaza el subtérmino siguiendo las siguientes reglas:

- Si a es una fracción entonces $a^n \rightarrow resultado\ de\ elevar\ la\ fraccion\ a\ la\ n$
- $(t^n)^m \rightarrow t^{resultado\ de\ n*m}$
- $(t1 * t2)^n \rightarrow t1^n * t2^n$
- $(t1 + t2)^n \rightarrow \Sigma_{i=0}^{n}(resultado\ de\ \begin{pmatrix} n \\ i \end{pmatrix})t1^{resultado\ de\ n-i}t2^i$

Es importante que en la implementación del binomio de Newton, si en $\binom{n}{i}t1^{n-i}t2^i$ se concatenan fracciones cuando los exponentes son 1, estas deben ser multiplicadas de una vez como producto de ejecutar `pow`. Por ejemplo si $n, i, t1, t2 = 2, 1, 2x4, 3y$ entonces $\binom{n}{i}t1^{n-i}t2^i$ sería igual a $2(2x4)(3y)$, sin embargo la función `pow` debe devolver para ese subtérmino $48xy$.

La función `pow` debe ser implementada de forma que resuelva todas las exponenciaciones posibles, de modo que si resolver una exponenciación genera otra exponenciación resoluble, entonces esta nueva exponenciación debe ser resuelta también con la misma llamada de `pow`.

3.4. Antiderivadas

En un módulo llamado Antidiff.hs usted debe implementar la función de integración simbólica `simbInt`, que dado un Término hace una búsqueda entre los subtérminos y cada vez que encuentra una integral, sustituye el subtérmino siguiendo las reglas descritas a continuación:

- Si a es una fracción, entonces $\int adx \rightarrow ax$
- Si a es una fracción, entonces $\int atdx \rightarrow a\int tdx$
- $\int t1 + t2dx \rightarrow \int t1dx + \int t2dx$
- $\int x^k dx = \frac{1}{k+1}x^{k+1}$
- Si x no ocurre en el término t, entonces $\int tdx \rightarrow tx$
- $\int sen(x)dx \rightarrow (-1)cos(x)$
- $\int cos(x)dx \rightarrow sen(x)$
- Si k es una fracción con denominador 1, entonces $\int sen(kx)dx \rightarrow \frac{-1}{k}cos(kx)$
- Si k es una fracción con denominador 1, entonces $\int cos(kx)dx \rightarrow \frac{1}{k}sen(kx)$
- $\int cos(x)sen^n(x)dx \rightarrow \frac{1}{n+1}sen^{n+1}(x)$
- $\int cos^n(x)sen(x)dx \rightarrow \frac{-1}{n+1}cos^{n+1}(x)$
- Si k es una fracción con denominador 1, entonces $\int cos(kx)sen^n(kx)dx \rightarrow \frac{1}{resultado\ de\ k*(n+1)}sen^{n+1}(kx)$
- Si k es una fracción con denominador 1, entonces $\int cos^n(kx)sen(kx)dx \rightarrow \frac{-1}{resultado\ de\ k*(n+1)}cos^{n+1}(kx)$
- Si n es un entero par entonces $\int sen^n(x)dx \rightarrow \int(\frac{1}{2}(1 + (-1)cos(2x)))^{resultado\ de\ \frac{n}{2}}dx$
- Si n es un entero impar entonces $\int sen^n(x)dx \rightarrow \int(1 + (-1)cos^2(x))^{resultado\ de\ \frac{n-1}{2}}sen(x)dx$

- Si n es un entero par entonces $\int cos^n(x)dx \rightarrow \int (\frac{1}{2}(1+cos(2x)))^{resultado\ de\ \frac{n}{2}} dx$
- Si n es un entero impar entonces $\int cos^n(x)dx \rightarrow \int (1+(-1)sen^2(x))^{resultado\ de\ \frac{n-1}{2}} cos(x)dx$
- Si n es un entero par entonces $\int sen^n(kx)dx \rightarrow \int (\frac{1}{2}(1+(-1)cos((2*k)x)))^{resultado\ de\ \frac{n}{2}} dx$
- Si n es un entero impar entonces $\int sen^n(kx)dx \rightarrow \int (1+(-1)cos^2(kx))^{resultado\ de\ \frac{n-1}{2}} sen(kx)dx$
- Si n es un entero par entonces $\int cos^n(kx)dx \rightarrow \int (\frac{1}{2}(1+cos((2*k)x)))^{resultado\ de\ \frac{n}{2}} dx$
- Si n es un entero impar entonces $\int cos^n(kx)dx \rightarrow \int (1+(-1)sen^2(kx))^{resultado\ de\ \frac{n-1}{2}} cos(kx)dx$

4. Programa Principal

Se desea que usted implemente un programa en Haskell, que cuando se escriba en ghci el comando `solve (Integ <termino> x)` se imprima un archivo de nombre `index.html` (con el formato descrito mas adelante), con todos los pasos para resolver la antiderivada de `termino`.

Para hacer esto, usted debe inicialmente aplicar la función `groupAtoms` a `termino`, para luego aplicar las funciones `simbInt`, `pow`, `distrib`, `groupAtoms` y `simplify` (en ese orden) y repetir estas últimas 5 operaciones, hasta que el termino resultante no tenga subtérminos con Integrales.

Adicionalmente, usted debe usar el Monad IO para ir imprimiendo en el archivo, los términos resultantes de cada una de estas funciones, en la medida que se van ejecutando. Cada resultado intermedio, debe ser impreso en el archivo, en una línea nueva que empiezan y terminan con los símbolos `$$`, pero si el resultado de una de estas funciones imprimiría una línea idéntica a la línea anterior, entonces esta línea no debe imprimirse.

Por ejemplo al escribir en ghci el comando `solve (Integ ((sen(x))^2) x)`, entonces imprime el archivo index.html de la siguiente forma:

```
<html>
<head>
<link rel="stylesheet" href="http://stilgar.ldc.usb.ve/Aledania/static/css/bootstrap.min.css" >
<script src="http://stilgar.ldc.usb.ve/Aledania/static/js/bootstrap.min.js"></script>
<script type="text/javascript" src="http://stilgar.ldc.usb.ve/Aledania/static/js/
mathjax-MathJax-v2.3-248-g60e0a8c/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>
</head>
<body>
$$\int sen^{2}(x)dx$$

$$=\int (\frac{1}{2}(1+(-1)cos(2x)))^{1}dx$$

$$=\int \frac{1}{2}(1+(-1)cos(2x))dx$$

$$=\int \frac{1}{2}+\frac{-1}{2}cos(2x)dx$$

$$=\int \frac{1}{2}dx+\int \frac{-1}{2}cos(2x)dx$$

$$=\frac{1}{2}x+\frac{-1}{2}\int cos(2x)dx$$

$$=\frac{1}{2}x+\frac{-1}{2}\frac{1}{2}sen(2x)$$

$$=\frac{1}{2}x+\frac{-1}{4}sen(2x)+C$$
```

```
</body>
</html>
```

5. MathJax

El programa principal descrito en la sección anterior crea un archivo html donde en el head del archivo se solicita un script llamado MathJax.js, este script es un traductor de latex a html escrito en javascript que se ejecuta justo cuando termina de cargar la página, de modo que si usted tiene conexión al servidor stilgar.ldc.usb.ve, podrá ver en el navegador, cuando cargue la página descrita en la sección anterior, lo siguiente:

$$\int sen^2(x)dx$$

$$= \int (\frac{1}{2}(1 + (-1)cos(2x)))^1 dx$$

$$= \int \frac{1}{2}(1 + (-1)cos(2x))dx$$

$$= \int \frac{1}{2} + \frac{-1}{2}cos(2x)dx$$

$$= \int \frac{1}{2}dx + \int \frac{-1}{2}cos(2x)dx$$

$$= \frac{1}{2}x + \frac{-1}{2}\int cos(2x)dx$$

$$= \frac{1}{2}x + \frac{-1}{2}\frac{1}{2}sen(2x)$$

$$= \frac{1}{2}x + \frac{-1}{4}sen(2x) + C$$

Usted también puede descargar previamente los recursos ubicados en el servidor stilgar.ldc.usb.ve y correr todo localmente si no posee conexión.

6. Condiciones de entrega

Si un programa no se ejecuta, el equipo tiene cero como nota del proyecto.

Se considerará para su evaluación los aspectos de modularidad y diseño del código Haskell.

El trabajo es por equipos de laboratorio. Debe entregar los códigos fuentes de sus programas, en un archivo comprimido llamado *Proyecto1-X-Y.tar.gz*, donde X y Y son los números de carnet de los integrantes del grupo. La entrega se realizará *antes* de la $1:00$ pm del $8-$ de Marzo de 2019 por aula virtual. El no cumplimiento de algunos de los requerimientos podrá resultar en el rechazo de su entrega.

7. Referencias

[1] *Wolfram Mathematica.*
https://www.wolfram.com/mathematica/

[2] *Maple for Academics.*
https://www.maplesoft.com/products/Maple/academic/

[3] *Wolfram Alpha Computational Intelligence.*
https://www.wolframalpha.com/

[4] *Symbolab Math Solver.*
https://www.symbolab.com/