Mathematical Expression Language

Language Proposal

COMS 4115 – Professor Edwards

Paresh Thatte – pat70@columbia.edu
Manjiri Phadke – mp3212@columbia.edu

Introduction

Mathematical notation is a language of symbols for representing patterns, restrictions and relationships of entities in the form of one or more equations of a set of variables. Equations are often represented in software systems. A small set of inputs can drive amortization schedule report, permission lists can trigger violation alerts on trade restrictions, and risk models can re-calculate pre-trading limits in real time.
Manipulating these equations by expansion, factorization, solving (simultaneously) and other symbolic rearrangement using (high-school) algebra rules can reduce the complexity of a problem. MEL provides a way to program these algebra rules to solve, simplify and manipulate in any way such common mathematical equations.

Overview

Mathematical Expression Language (MEL) is a programming language that lets a programmer define restriction rules or derivative relationships in the form of algebraic equations. The syntax allows you to express one or more equations using the standard algebraic notation. Equations of a valid form can be accepted and transformed by a programmer into a different form and other operations can be performed in them.

One transformation possible is to express the equation in terms of one variable, or solve for it given numbers. Other possible transformations are combining simultaneous equations, expanding a factorized expression or factorizing a simplified expression, collecting terms containing one of the variables in an expression.

Tokens

Basic math symbols: +, -, *, /, =, <, >, exp symbol - **
Separator: ,
Line Terminator: ;
Grouping: parentheses - (,)
Block delimiters – {, }
Line Comments: //
Block Comments: /*, */

Identifiers

String literals: Upper or lowercase character strings

Constants
Integer Literals: 0 - 9
Boolean: true, false

Operators
Basic math: +, -
Comparison: ==, !=, <, >, !
Assignment: =

Types
Collections - Array
Algebraic Types - Eq, Poly, Term, Sym, Coeff, Exp

Syntax
Function declaration - function
Conditional/Loop statements – if/then/else, for/while
Control keywords – return
Algebra keywords – lhs, rhs, terms, coeffs, exps, syms, num, den, (parts?)

Built-in functions
Term – hasSym, addSym, removeSym, print

Example program

A programmer could express “algebraic expansion" as a function in the RAL language, that is, you they write something like

```
expand((x+y)**2)
```

and it would return

```
x**2 + 2*x*y + y**2
```

Using loop constructs the binomial expansion algorithm could accept any binomial equation of the form

```
expand((a+b)**n)
```

and return all the terms in the summation, from k=0 to n, of
${}_nC_{k*}$(a**(n-k)*b**k)

Another example is a "solve" function that can be written to accept an equation and a target variable to solve for, in the form

```
solve(y, x + y = 0)
```

which would return

```
-x
```

Code example

```
function solve(Sym v, Eq e) {
        move(v, e.rhs, e.lhs);
        keep(v, e.lhs, e.rhs);
        print e.rhs;
}

function move (Sym v, Poly from, Poly to) {
        for(int i=0; i< from.terms.length; i++) {
                Term curTerm = from.terms[i];
                if(curTerm.hasSym(v)) {
                        to.coeffs[i] = to.coeffs[i] - from.coeffs[i];
                        to.terms[i].addSym(v);
                        from.terms[i].removeSym(v);
                        if(from.terms[i].length == 0) {
                                from.coeffs[i] = 0;
                        }
                }
        }
}

function keep (Sym v, Poly from, Poly to) {
        for(int i=0; i< from.terms.length; i++) {
                Term curTerm = from.terms[i];
                //move coeffs
                if(curTerm.syms.length > 1 || !curTerm.hasSym(v)) {
                        to.curTerm.coeffs[i] = to.curTerm.coeffs[i] – from.coeffs[i];
                }
                for(int j =0; j<curTerm.syms.length; j++) {
                        Sym curSym = curTerm.syms[j];
                        if(curSym != v){
                                to.terms[i].addSym(curSym);
                                from.terms[i].removeSym(curSym);
                        }
                }
        }
}
```