Mathematical Expression Language

Language Reference Manual

COMS 4115 – Professor Edwards

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

Contents

Introduction 2
Lexical Conventions 2
- Punctuation Tokens 2
  - Basic math symbols: +, -, *, /, =, <, >, exp symbol - ** 2
  - Separator: Period - . , Comma - , ... 2
  - Line Terminator: ; 2
  - Parameter Grouping: quotes – “,”, parentheses - (,), [,] 3
  - Block delimiters – {, } 3
  - Line Comments: // 3
  - Block Comments: /*, */ 3
- Identifiers 3
- Constants 3
  - Integer Literals: 0 - 9 3
  - Boolean: true, false 3
- Operators 3
  - Basic math: +, -, *, / 3
  - Comparison: ==, !=, <, <=, >, >=, !, &&, || 3
  - Assignment: = ... 3

Keywords 4
- Types 4
  - Collections - Array 4
  - Algebraic Types - Eq, Poly, Term, Sym, Coeff, Exp 4
- Syntax blocks 4
  - Function declaration - function 4
  - Conditional/Loop statements – if/else, for 4
  - Control keywords – return, continue 4
  - Algebra keywords – lhs, rhs, terms, coeffs, exps, syms, (parts?) 4
- Built-in functions 5
  - Term – hasSym, addSym, removeSym, print 5

Precedence and Associativity 5
Variable Scope 5
Statements 5
  - Basic statements 5
  - Statement Flow 5
  - Syntax blocks 6

Translation, Execution and Runtime 6

Introduction

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.

Lexical Conventions

Programs are written in an MEL file and translated as described in the last section.

Tokens accepted can be:

- Punctuation
- Constants
- Identifiers
- Operators
- Keywords

Keywords accepted can be for:

- Types
- Syntax blocks
- Built-in functions

Punctuation Tokens

Basic math symbols: +, -, *, /, =, <, >, exp symbol - **

These can be used in type declarations or as manipulation instructions.
In type declarations:
=, <, > are allowed to separate the Left and Right side of an equation.
+, - are allowed to separate terms in a polynomial
*, ** are allowed within a term. The exponential token is only allowed with symbols.

As instructions:
+, -, *, / are allowed with integers, and = is allowed with identifiers

Separator: Period - . , Comma - ,

Commas are allowed to separate parameters in a function declaration or call.
Periods are allowed to reference parts of algebraic types.

Line Terminator: ;

These are allowed as termination tokens for statements

Parameter Grouping: quotes – “,”, parentheses - (,), [,]

These are allowed in type or function declaration, and in array access instructions.

Block delimiters – {, }

These are allowed as function block or logical block delimiters

Line Comments: //

These are allowed for entering comments on the line. All tokens on the line are ignored.

Block Comments: /*, */

These are allowed for entering a block of comments. All tokens within the block are ignored.

Identifiers

String literals: Upper or lowercase character strings

Identifiers are allowed in assignments to Types, constants and functions. All string literals except keywords can be used as identifiers.

Constants

Integer Literals: 0 - 9

Integer literals are allowed as constants such as a coefficient of a term, or as values of identifiers such as a loop counter etc.

Boolean: true, false

Boolean constants are allowed in conditional statements, as function return values, as identifiers for either purposes etc.

Operators

Basic math: +, -, *, /

Basic math operations are allowed with integer literals or identifiers referring to integer literals.

Comparison: ==, !=, <, <=, >, >=, !, &&, ||

Logical comparison operations can be performed on Boolean constants, identifiers referring to Boolean constants or the results of instructions that produce a Boolean constant. These are allowed in conditional statements.

Assignment: =

Assignment tokens are allowed with identifiers.

Keywords

Types

Collections - Array

Arrays can be constructed of any Types or constants, and can be assigned to identifiers.

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

These are custom structures that are defined by the language for simplifying access to the individual parts of the algebraic notation being manipulated.

Syntax blocks

Function declaration - function

All MEL programs are organized within functions. The function keyword is followed by the return type, the identifier for the function name, and the parameter arguments list in parenthesis, followed by the statements block enclosed in the block delimiters.

Conditional/Loop statements – if/else, for

The looping construct consists of the keyword followed by - the loop identifier assignment, termination condition, and stepping instruction - in parenthesis, followed by a block of statements enclosed in the block delimiters.

The conditional keywords are followed by the comparison operator in parenthesis, followed by a block of statements within the block delimiters.

Control keywords – return, continue

The return keyword breaks the function block and returns immediately, while the continue keyword breaks the current instance of the looping construct.

Algebra keywords – lhs, rhs, terms, coeffs, exps, syms, (parts?)

The two parts of an equation can be accessed using the lhs (left-hand side) and rhs (right-hand side) keywords. Each part is of type Poly.

The three types of parts in a Poly can be accessed using the keywords coeffs, terms, and exps. Each returns an array indexed by the highest power symbol in that position. The coeffs returns an integer literal array, while the other two in turn return arrays – the symbols within each term, and their respective exponents.

Built-in functions

Term – hasSym, addSym, removeSym, print

The symbol manipulation functions support adding or removing a symbol from a given term. The default exponent is the current array index.

The print function can be used with any constants or types associated with an identifier.

Precedence and Associativity

An Algebraic Type's part's access has the highest precedence.

Math operations follow the normal precedence and associativity rules:

*, /
+, -

Comparison operators have higher precedence than logical combination operators in conditional blocks:

!
==, !=, <, <=, >, >=
&&, ||

Assignment operators have lower precedence than type declarations and keywords.

Variable Scope

An MEL program is organized within functions. There are no global variables, all variables are local to the function they are declared in.

The function parameters are considered local variables, and have the same scope.

The scope of Loop identifiers is within the declaring loop only.

Statements

Statements are present within function bodies and are terminated by the semi-colon token.

Basic statements

These are operations such as Type declarations, Identifier assignment, Type modification, Math operations, function calls and flow control instructions. These have to end in a termination token.

Statement Flow

Multiple statements can be listed within a function and they must be separated with semicolons. All statements will be evaluated in the order listed and will be affected by the instructions declared in the previous statements.

Syntax blocks

Function declarations, conditional blocks and loop constructs can be listed followed by block-delimiter tokens.

Translation, Execution and Runtime

MEL programs must have a main function. The flow of the program is directed by the main function. The program source files are compiled and translated into java source files which are built into executables using the java compiler.