Language Documentation of dml

Created by Matthew Moltzau, with the help of BNFC

May 3, 2018

Introduction

'dml' stands for 'declarative matrix language' and is meant to be a tool for operations on 2D matrices. The specification is somewhat incomplete since many matrix operations have not been included so far.

This project was only possible thanks to BNFC, which generated many useful files including a lexer, parser, and abstract syntax tree.

Language Features

Variables

Integers, doubles, and matrices are the only types in dml. All types are interpreted in a manner similar to python, making dml an interpreted language.

Mathematical Operators

These include + - / * on both doubles and integer types. Addition and subtraction may be performed on rows, where a row is a subscripted matrix like so: A[i].

Rows have division and multiplication with scalars, and even have a swap operator '<−>', which means all standard row operations can be represented.

Output

Standard output is accessed via using the '=>' operator on 'out', while using 'tex' instead generates output in a LaTeX file. You will likely need to still edit any tex output, but it can at least automate creating matrices in .

ref and rref

The only implemented matrix operations as of now are ref and rref, which build on some of the row operations. Reduced-row echelon format doesn't work well with integers, so I also implemented casting.

Casting allows rref to work with doubles, then cast back to an integer, however, matrix casting can only be done explicitly.

Comments
Single-line comments begin with `//`.
Multiple-line comments are enclosed with `/*` and `*/`.

Implementation Challenges

One of the particularly difficult features to implement was allowing a matrix to hold both doubles and integers. c doesn't have a strong support for generic types, and I would have to code some sections differently, depending on the type being used.

The Syntactic Structure of dml

Non-terminals are enclosed between ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

⟨*Program*⟩ ::= ⟨*ListStm*⟩

⟨*Stm*⟩ ::= `create` ⟨*String*⟩ `titled` ⟨*String*⟩ `for` ⟨*String*⟩
| ⟨*Ident*⟩ => `tex`
| ⟨*String*⟩ => `tex`
| `close tex`
| `out` => `tex`
| ⟨*Ident*⟩ => `out`
| ⟨*String*⟩ => `out`
| ⟨*Row*⟩ => `out`
| ⟨*Ident*⟩ `::=` ⟨*ListRow*⟩
| ⟨*Row*⟩ <-> ⟨*Row*⟩
| ⟨*Row*⟩ -> ⟨*Row*⟩
| ⟨*Row*⟩ = ⟨*Row*⟩
| ⟨*Ident*⟩ => `rref`
| ⟨*Ident*⟩ => `ref`
| ⟨*Ident*⟩ = ⟨*Exp*⟩
| ⟨*Ident*⟩ => `double`
| ⟨*Ident*⟩ => `int`
| `pause`

⟨*Row*⟩ ::= `[` ⟨*ListExp*⟩ `]`
| ⟨*Row*⟩ + ⟨*Row1*⟩
| ⟨*Row*⟩ − ⟨*Row1*⟩
| ⟨*Row1*⟩

⟨*Row1*⟩ ::= ⟨*Exp2*⟩ `*` ⟨*Row2*⟩
| ⟨*Row2*⟩ `/` ⟨*Exp2*⟩
| ⟨*Row1*⟩ `*` ⟨*Exp2*⟩
| ⟨*Row2*⟩

⟨*Row2*⟩ ::= ⟨*Ident*⟩ `[` ⟨*Exp*⟩ `]`
| `(` ⟨*Row*⟩ `)`

⟨*Exp*⟩ ::= ⟨*Exp*⟩ + ⟨*Exp1*⟩
| ⟨*Exp*⟩ − ⟨*Exp1*⟩
| ⟨*Exp1*⟩

⟨*Exp1*⟩ ::= ⟨*Exp1*⟩ `*` ⟨*Exp2*⟩
| ⟨*Exp1*⟩ `/` ⟨*Exp2*⟩
| ⟨*Exp2*⟩

⟨*Exp2*⟩ ::= ⟨*SInteger*⟩
| ⟨*Double*⟩
| ⟨*Ident*⟩
| `e`
| `pi`
| `(` ⟨*Exp*⟩ `)`