EZMath Final Report

Piaoyang Cui[1], Yi Wang[2], Shangjin Zhang[3], and Zhejiao Chen[4]

[1]pc2618@columbia.edu, Columbia University
[2]yw2580@columbia.edu , Columbia University
[3]sz2425@columbia.edu, Columbia University
[4]zc2291@columbia.edu, Columbia University

December 17, 2014

Contents

1 Introduction **1**

2 Language Tutorial **2**
2.1 Greatest Common Divisor(GCD) 2
2.2 Variables 5
2.2.1 Float 6
2.2.2 Matrix 6
2.3 Unit-Operator 6
2.4 Logical Validation 7
2.5 Cumulative Sum and Product 7
2.6 Formulas 7

3 Language Reference Manual **9**
3.1 Types 9
3.1.1 Float 9
3.1.2 Matrix 9
3.1.3 Supported Matrix Operators 10
3.2 Identifiers 10
3.3 Variables 10
3.4 Keywords 11
3.5 Binary and Unit Operators 11
3.5.1 Float Number Operators 11
3.5.2 Logical Operators 11
3.6 Separators 11
3.7 White Space 12
3.8 Formula 12
3.8.1 Formula Definition 12
3.8.2 Formula Parameters 12
3.8.3 Return Value 13
3.8.4 Formula Evaluation 13
3.8.5 Recursive Formula 13
3.8.6 Piecewise Formula 14
3.8.7 Nested Formula 14

3.9 Logical Expressions . . . 14
3.10 Statement . . . 15
3.11 Comments . . . 15
3.12 Input Program Structure and Scope . . . 16
3.12.1 Program Structure . . . 16
3.12.2 Scope . . . 16
3.13 Output Program Structure and Scope . . . 16
3.13.1 Program Structure . . . 16
3.13.2 Scope . . . 17

4 Project Plan **18**
4.1 Project Process . . . 18
4.1.1 Planning . . . 18
4.1.2 Specification . . . 18
4.1.3 Development Environment . . . 19
4.1.4 Testing . . . 19
4.2 Programming Style Guide . . . 20
4.3 Project Timeline . . . 20
4.4 Roles and Responsibilities . . . 22
4.5 Development Environment . . . 22
4.6 Project Log . . . 22

5 Architectural Design **23**
5.1 Overview . . . 23
5.2 Scanner . . . 24
5.3 Parser . . . 24
5.4 Abstract Syntax Tree . . . 24
5.5 Interpreter . . . 24
5.6 Compiler . . . 25

6 Test Plan **26**
6.1 Automated Testing . . . 26
6.2 Test Cases . . . 28

7 Lessons Learned **30**
7.1 Voices of the Members . . . 30
7.2 Advice for Future Teams . . . 32

8 Appendix **33**
8.1 Project Log . . . 33
8.2 Source Code . . . 43

Chapter 1

Introduction

Complex mathematical operations and representations are always highly demanded for scientific programming. When writing academic papers, LaTeX, a markup language to typeset document, is often used to prettify mathematical expressions and the overall layout. By adopting syntax from LaTeX, user can easily type complicated mathematical equations for calculation purpose. Thus, we propose a new programming language called `EZMath` written completely in LaTeX syntax.

The targeted usage of this language can be described in the following scenario: A top-notch mathematic paper involving a substantial amount of complicated math expressions and functions along with text is written purely in LaTeX (i.e. paper.tex), thus it can be compiled by LaTeX compiler to a beautifully and smoothly typed pdf file (paper.pdf). Futhermore , taking in the same source file paper.tex, compile it through the `EZMath` compiler, the output file is a C++ source file(paper.cpp) which can further be compiled by a C++ compiler to generate an executable file. The C++ source file translates every valid formula definition into a function. While the main() function directs a complete report of the `EZMath` compiling process to an output file report.tex in LaTeX syntax. We also provide an interpreter along with the compiler that can interpret paper.tex to a user-friendly command-line report output, which contains the same information of final result.pdf.

Chapter 2

Language Tutorial

2.1 Greatest Common Divisor(GCD)

We'd like to begin presenting `EZMath` with well-known GCD (Greatest Common Divisor) algorithm. Suppose we have follow GCD program in gcd.tex file

```
...
$$
gcd(a,b) =
    \begin{cases}
        a & a == b \\
        gcd(a-b,b) & a > b  \\
        gcd(a, b-a) & b > a
    \end{cases}
$$
...
$$
    gcd(9,21)
$$
...
```

As you can see, we defined a formula called gcd and want to know the gcd of 9 and 21. Also you should notice that, we use $$ to wrap the formula definition and call. This is necessary because `EZMath` only focus on math formula parts of gcd.tex and ignore texts.

Type `make` and let the OCaml generates `EZMath` program, there're several options to get the final expected result (or summary).

One is interpreting:

```
EZMath -i gcd.tex
```

This command uses our inside interpreter program just as `microc` does, which will generate the output in the stdout:

```
title{(No Title)} author{Unknown Author}
----------------------------------------
         Formular Definitions
----------------------------------------
gcd(a, b) = {
    a, if a==b. Or
    gcd(a-b, b), if a>b. Or
    gcd(a, b-a), if b>a.
}
----------------------------------------
         Formular Evaluation
----------------------------------------
gcd(9,21) = 3
----------------------------------------
         Logical Validation
----------------------------------------
----------------------------------------
         Variable Definitions
----------------------------------------
----------------------------------------
         Matrix Definitions
----------------------------------------
```

Our program takes the `gcd.tex`, creates a formula symbol table for `gcd`, then takes `gcd(9,21)` as execution statement, through a series of computation, finally it will output all the information. Because there's no information about Logical Validation, Matrix Definitions, these areas are left blank. We will introduce them later.

Another is compiling:

```
EZMath -c gcd.tex
```

This command uses our inside compiler, which will generate a cpp file (by default, named as result.cpp):

```
...
double gcd(double a, double b){
  if(a==b) return a;
  if(a>b) return gcd(a-b, b);
  if(b>a) return gcd(a, b-a);
  throw std::runtime_error("Illegal parameter in piecewise
      function gcd");
}

int main(int argc, char ** argv) {
...
```

```
c_result[0]=string("")+"gcd(9, 21)=gcd("+dtos(9)+", "+dtos
    (21)+")="+dtos(gcd(9, 21));
...
return 0;
}
```

After we get the result.cpp, we can further compile it into executable file by

```
g++ -o -std=c++11 result result.cpp
```

And then we execute it. Notice, sine we take advantage of new feature came with C++ 11, λ functions, to meet certain mathematical needs like $\sum$ and $\prod$. However in this case, `-std=c++11` is just optional. We recommended using it when you feel the needs to implement some advanced math features.

```
./result
```

Now we will get a new LaTeX file called result.tex (by default), as the summary of the input tex file.

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{amsmath}
\title{(No Title)}
\author{Unknown Author}
\date{December, 2014}
\begin{document}
\maketitle
\section*{Variables Definition}
\section*{Matrix Definition}
\section*{Formula Definition}
\begin{gather*}
gcd(a, b)=\begin{cases}
a & a==b\\
gcd(a-b, b) & a>b\\
gcd(a, b-a) & b>a
\end{cases}
\end{gather*}
\section*{Logical Validation}
\section*{Formula Evaluation}
\[
gcd(9, 21)=gcd(9, 21)=3
\]
\end{document}
```

You can compile and display result.tex with your own LaTeXcompiler.

And finally, something like Figure 2.1 will get displayed in your final pdf file.

As we can see, we get a similar summary as interpreter. But it looks much pretty!

(No Title)

Unknown Author

December, 2014

Variables Definition

Matrix Definition

Formula Definition

$$gcd(a,b) = \begin{cases} a & a == b \\ gcd(a-b,b) & a > b \\ gcd(a,b-a) & b > a \end{cases} \quad (1)$$

Logical Validation

Formula Evaluation

$$gcd(9,21) = gcd(9,21) = 3$$

Figure 2.1: Displayed result.tex

Through this simple example, we can see that `EZMath` is capable of processing piecewise formula definition and call, as well as generating summary in two ways. However, we can do more! Let's meet other interesting features in the following sections.

2.2 Variables

We have two types of variables: float and matrix.

2.2.1 Float

Float is a representation of single decimal number. You can assign a name to it and use it as float variable. Note that, we consider any integer as decimal number in `EZMath`.

We use a parentheses pair surrounding negative number to avoid reduce/reduce error during the parser step, it increases a little overhead, but eliminate the use of ; separator (As in the C language).

Example:

```
$$a1 = (-2)$$
$$a2 = 2 ^ {a1}$$
$$a3 = a1 + 1.5$$
```

2.2.2 Matrix

Matrix is a representation of a block of decimal numbers. Note that, matrix can only contain simple decimal numbers and negative number is not allowed during the definition. You cannot use a float variable to construct a matrix. You can also assign a unique identifier to a matrix and then it becomes a matrix variable. Further information can be referenced at 3.1.2

Example:

```
$$m1 = \begin {bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
1 & 1 & 1
\end {bmatrix}$$

$$m2 = 2 * m1$$
$$m3 = m1 ^ {T}$$
$$m4 = m1 * \begin {bmatrix}
            1 \\
            1 \\
            1
            \end {bmatrix}$$
```

2.3 Unit-Operator

Besides simple arithmetic operations such as +, -, * and /, we have some build-in unit-operations.

Example:

```
$$ b1 = \sin {\pi/2}$$
$$ b2 = \log {10 ^ {2}}$$
```

Note that, keyword
pi will be replaced by 3.1415926..., e is recongnized as natural logarithm, and log operation use base 10 as default.

2.4 Logical Validation

Logical validation allows user to valid some true or false statements. For example:

```
$$c1 = 3 c2 = 4 c3 = 5$$
$$c1 ^ {2} + c2 ^ {2} == c3 ^ {2}$$
```

Note that, there is no special separator between any two expressions. Some spaces is enough. However, wrapping unrelated expressions with $$ is recommended.

2.5 Cumulative Sum and Product

We also support advanced math operations such as cumulative sum and product. For example:

```
$$sum(p) = \sum _ {i=1} ^ {p} {i}$$
$$prod(p) = \prod _ {i=1} ^ {p} {i}$$

$$sum(100)$$
$$prod(4)$$
```

You can even use these operations to implement some complicated loops.

Note that, in order to use cumulative sum and product, you should use `-std=c++` option to compile the cpp file.

```
g++ -o result result.cpp -std=c++11
```

2.6 Formulas

Recall that gcd() is a piecewise formula because it contains several expressions with their conditions. Also we support basic expression defined in formula, we call it **regular expression**, and call the former one **piecewise expression**.

`\sum_{}^{} and \prod_{}^{}`

are special regular formulas with single expression, which reduce the overhead of writing loops.

Moreover, you can define recursive formula.

The famous fibonacci number can be written as follows,

```
$$fib(x) = \begin{cases} fib(x-1) + x & x>0 \\
                         1 & x==0
           \end{cases}
$$

$$fib(5)$$
```

To build recursive a formula, you will always follow piecewise formula because you always need a termination condition.

The famous pythagorean theorem could be written as follows,

```
$$
pt(x,y,z) = x ^{2} + y ^{2} == z ^{2}
$$
```

Also, `EZMath` supports formula overloading, which means you can define two same name formula with different signatures.

```
$$sum(low, high) = \sum _ {i=low} ^ {high} {i}$$
$$sum(1, 100)$$
$$sum(100)$$
```

More information about formula regarding its parameter, return value, etc, could reference the section 3.8 in Language Reference Manual.

Chapter 3

Language Reference Manual

3.1 Types

3.1.1 Float

By default, all the numbers including integer and decimal numbers are recognized as float type number in OCaml.

Due to the speciality of `EZMath`, all the decimal numbers appear in the expression except those as matrix elements are only allowed in the format like (-Float), inside a pair of parentheses. However, negative number are supported as matrix elements.

3.1.2 Matrix

Matrix constants are matrix literals in the code. The type of the elements of the matrix is number constant. Matrix constant is declared by LaTeX matrix grammar.

`\begin{bmatrix}...\end{bmatrix}`

Some examples:

$$\begin{bmatrix}10.5 & 20.2 & 30.5\end{bmatrix} \quad \begin{bmatrix}5.2\\6.1\\7.3\end{bmatrix} \quad \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix} \quad \text{m} = \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

3.1.3 Supported Matrix Operators

`EZMath` supports plus, minus, multiply, dot multiply and transpose ($+, -, \times, \cdot, ^T$) as matrix operators. Operands can be matrix-type variables or matrix constants. Plus, minus, multiply and dot multiply can be applied between two matrices, but the sizes of the operands must agree on the requirements of matrix's operation. In addition, multiply can also be applied between a float number and a matrix (the order is mandatory). Transpose can be applied on a single matrix. The result of matrix operators will be a matrix-type value.

Operator	Definition	Example
+	matrix add	A + B
-	matrix minus	A - B
\times or *	float/matrix multiply	A \times B
\cdot	multiply the corresponding elements	A $\cdot$ B
^{T}	matrix transpose	A^T

3.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. Upper and lower case letters are considered different. Characters besides letters and digits are not allowed in identifiers, including underscore (_) and hyphen (-).

Declaring two identifiers with the same literal or changing the definition of a previously declared identifiers is not allowed. Compiler should report error on such attempts. An exception is function identifier: functions can have same identifier as long as they have different number of parameters (called signature), as known as function overloading. See function section for more detail.

3.3 Variables

Variables in `EZMath` should be defined in

```
$$
%something
$$,
```

e.g.

```
$$ x = 2 $$
$$ a = 3, b = 4, c = 5$$
```

By default, any variable defined is of type Float for the ease of arithmetic calculations.

3.4 Keywords

e	The natural logarithm approximately equals to 2.71828.
\pi	π approximately equal to 3.1415926.
sin	Reserved for the \sin function
cos	Reserved for the \cos function
tan	Reserved for the \tan function
log	Reserved for the \log function

3.5 Binary and Unit Operators

3.5.1 Float Number Operators

`EZMath` supports plus, minus, multiply, and divide (+, -, *, /) as basic binary arithmetic operators. These operations follow the conventions in languages like C and Java. Basic arithmetic operators can be applied between float-typed variables or number constants. The result will be a float-type value.

`EZMath` supports a subset of LaTeX's original math symbols, including:

$$\sin, \cos, \tan, \log$$

3.5.2 Logical Operators

`EZMath` supports the following logical operators: larger, less than, equal, larger or equal, less or equall, inequal (>, <, =, >=, <=, ! =). Logical operators can be used in conditions of piecewise formulas and logical expressions.

3.6 Separators

A separator separates syntax elements. `EZMath` has four separators (( ) , & \\). White space (see next section) is a separator, but it is not a token. The other separators are all single-character tokens themselves:

Separator	Usage
(White space)	Separates tokens (see next section)
,	Separates expressions in statement
&	Separates expression and condition in piecewise function
\\	Separates cases in piecewise function

3.7 White Space

White space is the collective term used for several characters: the space character, the tab character, the newline character, and the carriage return character. White space is ignored (outside of string and character constants), and is therefore optional, except when it is used to separate tokens. As a result, indents are inessential in `EZMath`.

3.8 Formula

A formula is a procedure of computations upon given arguments. It's essentially the same as the function in functional programming language. Since `EZMath` compiler only operates on text between two $$, whenever defining or calling a formula, embed the whole statement inside two $$.

3.8.1 Formula Definition

A formula definition contains a name (identifier), a pair of parentheses, and an optional list of parameters in the parentheses, an equal sign, and an expression, from left to right orders. Functions with same name, but different number of parameters are considered different functions, as known as function overloading.

Examples:

```
$$g() = 3$$
$$Sin(x) = \sin {x}  $$
$$Sin(a, b) = \sin {a + b} $$
```

Formula can refer variables other than parameters, as long as they are declared outside the formula. However, assigning variable is not allowed in formula definition. This also applies on the recursive, piecewise and nested formula that introduced below. In general, a formula should not have any effect of outside status.

3.8.2 Formula Parameters

A formula can either has no parameters or has any number of parameters of float type. The name convention of parameters is the same as identifiers, except parameters only need to be unique within the corresponding function definition.

Note that, it's not allowed to use the same name in parameter.

3.8.3 Return Value

Formula evaluations always return a float type value.

3.8.4 Formula Evaluation

A formula can be evaluation as follows:

```
$$g()$$
$$Sin(g())$$
$$Sin(8.0, 3)$$
```

Note that, we can use the result of a formula ($g()$) as the argument of another formula $Sin()$.

`EZMath` will recognize these formula calls and evaluate them, if the name of formula does exist, the number of arguments is correct, and the type of arguments are all float. Otherwise, an error will be reported by compiler.

`EZMath` follows applicative-order evaluation.

If the user explicitly calls a standalone formula with valid argument, e.g.

$$f(5)$$.

Then the main() function in generated C++ code will print out the returned value of type float as well as the values of global variables that have been used in this formula. Otherwise, the returned value will only be used in evaluation, and will not be printed.

Generally, you may get something like this in the final summary pdf:

Sin(g())=Sin(3)=0.14112

3.8.5 Recursive Formula

A formula represents a function, thus it's intuitive to support recursive formula. The usage of a recursive formula is illustrated as follows:

```
$$
r(x) = 2 * r(x/2)
$$
```

However, we should use piecewise formula to define the termination conditions for recursive formulas. Mutual-recursion is also supported (see the Nested Formula section for more detail).

3.8.6 Piecewise Formula

`EZMath` supports piecewise formulas. The usage of a piecewise formula is illustrated as follows:

```
$$
fac(x) = \begin{cases} fac(x-1)*x & x>0 \\
                       1          & x==0
         \end{cases}
$$
```

As shown above, we use \\to seperate cases. And in a particular case, we use & to seperate expression and condition.

3.8.7 Nested Formula

Nested formula means referencing another formula either in the definition of a formula. The reference can appear in both expressions and conditions. The formula declaration order is not important, so a formula can refer other formulas declared afterwards. The usage of a nested formula is illustrated as follows:

```
$$
bar(x) = x + 1
foo(x) = bar(x) * 2
$$
```

3.9 Logical Expressions

Logical expressions are supported in LaTeX as following:

$$3^2 + 4^2 == 5^2$$

`EZMath` can evaluate such expression and return a value of 1 (true) or 0 (false). This return value can be used in further evaulation. If the logical expression appears at the top-level, `EZMath` compiler will report the correctness for this logical expression in the report.tex.

$$a^2 + b^2 == c^2$$

For the above expression, `EZMath` compiler will check if a, b, c are defined and assigned values. If not, it will report error and stop compiling. Otherwise, it will check if the equality satisfies. If it does, it will report true in report.tex, false otherwise.

$$a < b$$

Similarly, `EZMath` compiler will check if a, b are defined and assigned values. If not, it will report error in report.tex. Otherwise, it will check if the inequality satisfies. If it does, it will report true in report.tex, false otherwise.

If the elements of an logical expression are all constants, `EZMath` will remember this logical expression and validate its correctness in report.tex.

If any of the elements is an identifier, it must be defined and assigned earlier. The compared values from left and right hand side of the logical operator are of type float only. Matrix is not allowed in any kind of logical expression. If an invalid logical expression is encountered, the `EZMath` compiler will report error and stop compiling.

3.10 Statement

A statement is a sequence of expressions, separated by comma (,). The range of lawful expression includes variable assignments, logical expressions and formula calls, etc. e.g.

```
$$ a = 2, b = 3, c = 4$$
```

```
$$ a < 2, c > 3, b == Sin(a, b), pt(3,4,5)$$
```

Normal derived expressions are also allowed in statement, e.g.

```
$$ (a + b),  (Sin(2, 3) * g()) - 1$$
```

However, single ID or float / matrix constant is not acceptable as expression in statement. Multiple statements can be put in a single $$ ... $$ block, EZMath will separate them automatically. e.g

```
$$ a = 2, b = 3 x = Sin(a, b)$$
```

a = 2, b =3 is the first statement and x = Sin(a, b) is the second.

3.11 Comments

In accordance with the LaTeX syntax for comments, everything after the % character until the end of the line is comment and is ignored by `EZMath` compiler.

3.12 Input Program Structure and Scope

3.12.1 Program Structure

The input file of `EZMath` should be a LaTeX file. The compiler of `EZMath` only operates on statements between the pair of $$ symbols. It ignores all other text outside the scope of math mode. It only supports the basic math typesetting, and only supported grammars are allowed.

Users who want to display only LaTeX math statements can use \[and \] instead. Statements between \[and \] will be ignored by `EZMath` compiler.

To sum up, `EZMath` Compiler will detect following structures:

1. Definition of Formulas
2. Statement, includes

   Top-level calling of formulas

   Top-level logical expressions

   Other expressions

3.12.2 Scope

Variables (float/matrix) and formula definitions are globally declared and globally visible in `EZMath`. Formula's scope can refer global variables, but can't assign or create any variable.

The order of declarations is flexible. Formula can refer variables and other formulas that declared anywhere in program, and a statement can call a function that declared anywhere, even in the end of the program. However, compiler will check each actual formula call for the existence of the referred global variables. If any referred variable doesn't exist, compiler will report error.

3.13 Output Program Structure and Scope

3.13.1 Program Structure

Basically, `EZMath` will output a .cpp file corresponding to the input LaTeX file. The C++ program contains several math functions and one main functions. Math functions correspond to each formula defined in the input LaTeX file, the main function is used to calculate and output the expression in the LaTeX file.

This c++ file includes several parts:

1. Declaration of global variables
2. Declaration of functions
3. Build-in matrix class
4. Definition of functions
5. Main function
6. Validations of logical expressions
7. Results of the formula calls
8. Outputting summary.

Compile and execute the .cpp ouput file will further generate a LaTeX file. The LaTeX file is a summary of the declarations and computations for the original input file. It will contain:

1. The basic information about the original input (title, arthors, etc)
2. All user defined variables and matrices
3. The validations of logical expressions
4. The definitions of formulas and
5. The results of the computations.

3.13.2 Scope

All the variables and formulas defined in the input file are global visible in the output cpp file.

Because we only output single cpp file without any header file, functions in cpp cannot be used in other cpp file. However, user can manually modify this cpp file to use such functions.

The LaTeX file generated by cpp is only for displaying purpose. There is no $$ symbol in it, so if you use this LaTeX file as a new input, you will get nothing. Instead, we use \[and \] to wrap formulas.

Chapter 4

Project Plan

Good planning is very essential for large projects like compiler to deliver on time. While looking back, we found some principals were established to facilitate the project management and delivery. This section identifies these principals.

4.1 Project Process

4.1.1 Planning

After we created our Language Reference Manual, we held two meetings to discuss the future work. We separated the milestones of the project and set up the expected complete date of each milestone. Later on we constantly reviewed and updated our plan according to the development by online discussions on Slack and offline meetings.

4.1.2 Specification

Language Reference Manual established a working draft of the specification of `EZMath` but we still kept the specification open for discussion. During development, we tried to implement the features defined in LRM in the order of importance, and if doubts and objections raised in development, we held discussions about whether we should modify or remove any feature. We also allow new features to enter the LRM, as long as it fits the overall goal of our languages, and members agree on the change. Language Guru was usually leading such kind of discussion.

4.1.3 Development Environment

Several tools have played great roles in our development: We use Github to store and synchronize our code, use Github issue tracker to track the bugs/defects, and use Slack to hold all the private and public discussion.

Slack has been proved great value in our development: we eliminated the use of emails/IMs completely and improve efficiency while working. Team communication become smoother than ever, with its great github, google docs integration and different channels for topics.

Additionally, we used ShareLaTeX to concurrently edit our project proposal, LRM and final report.

4.1.4 Testing

There are in general two phases for testing. The first phase is to test if all the specified language features in LRM are implemented correctly, and if all the exceptions can be caught and raised. The second phase comes into play after all the components are glued together. In this phase, we come up with corner cases that contain exceptions that can not be caught by the compiler or cases that produce unexpected outcome. Mostly, these cases arise due to the ambiguity from LRM, corner case negligence during implementation, and oblivion of implementation.

Regression Test

Regression test is done by running shell scrip to automatically run through all the previously passed test cases again whenever new feature is added, old feature is modified, or whenever needed. This is to make sure that the newly added or modified feature or any critical code change will not break the previously working feature.

Unit Test

Unit test is achieved by only testing a specific feature in one test case. Though unit test might not seen as important, it serves as great building block that builds up the entiring compiler.

4.2 Programming Style Guide

Space	Put space after punctuations.
Comment	Write comment for each function group and important declaration. Nested comment not allowed in code.
Code Reuse	Write utility functions for commonly reused code. Put common utility functions in header.ml file.
Value Declaration	Non-sensical or simple name not allow in declaration, e.g. a, b, output. Encourage anonymous function if possible, avoid naming pollution. Capital letter not allowed in value declaration and function declaration. Use underscore to separate words in names.

4.3 Project Timeline

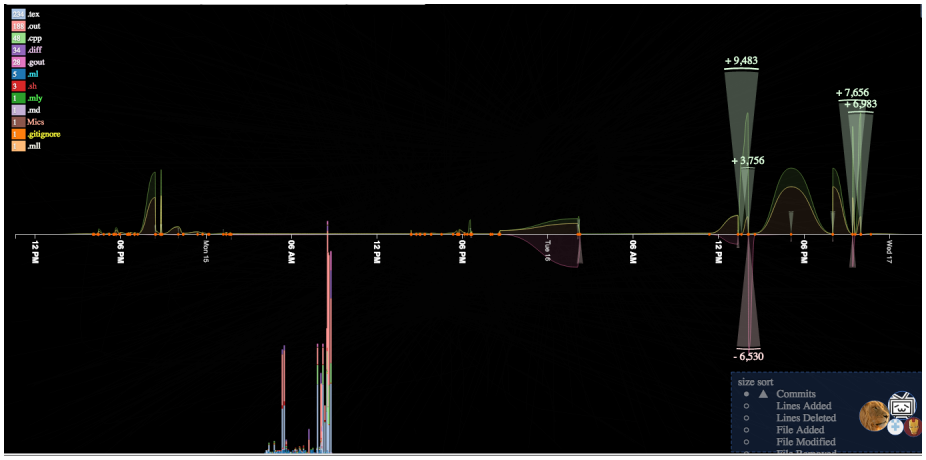

Figure 4.1: Overall Statistics

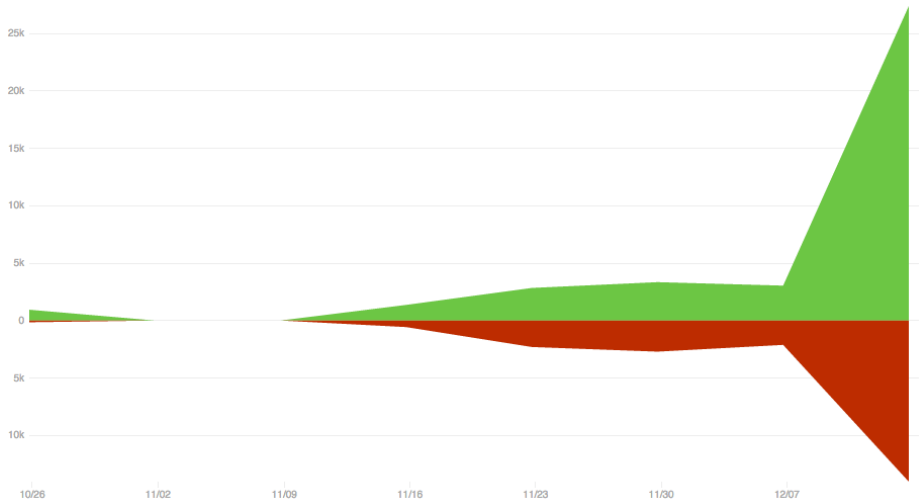

Figure 4.2: GitHub Additions & Deletions

Figure 4.3: GitHub Commits

4.4 Roles and Responsibilities

Name	Role	Responsibilities
Piaoyang Cui	Manager	Set up project plan and schedule. Break up tasks and assign to team members. Organize team meetings and discussions. General contribution to project (design, code and test).
Yi Wang	Language Guru	Lead the design the language specs and features. Decide detail of language implementation. General contribution to project (design, code and test).
Shangjin Zhang	System Architect	Lead the design of interpreter and compiler architecture. Lead the prototyping of `EZMath`. General contribution to project (design, code and test).
Zhejiao Chen	Tester	Create test plans and test suites. Write feature and regression tests for different components. Blackbox testing of the system. General contribution to project (design, code and test).

4.5 Development Environment

OCaml Compiler	version 4.01.0
C++ Compiler	GNU C+ 4.8, with C++11 surpport
Build Tool	GNU Make
Text Editor	Sublime Text
Team Collaboration	Slack
Version Control	Git (github.com)
Online LaTeX collaboration writing	sharelatex.com
Online C++ collaboration coding	coderpad.io

4.6 Project Log

We use statistics (those commits) from Github as our project log, details are located in the Appendix 8.1. Usernames have been replaced by real names.

Chapter 5

Architectural Design

5.1 Overview

`EZMath` combines compiler and interpreter. The architectural design follows the MicroC compiler[1].

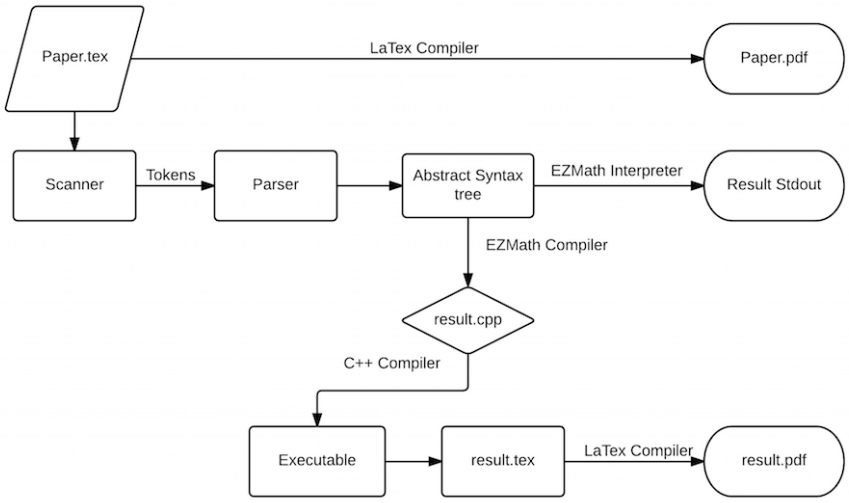

Figure 5.1: Architecture

The `EZMath` compiler transforms a LaTeX file into a c++ program. The cpp file generated by `EZMath` compiler can further be compiled by available c++

[1] http://www.cs.columbia.edu/ sedwards/classes/2014/w4115-fall/microc.tar.gz

compiler and then output another LaTeX file. This output LaTeX file can further be compiled into a pdf file, which is a summary of the input LaTeX file.

Moreover, the `EZMath` interpreter prints out a readable summary of the input LaTeX file directly.

Both the compiler and interpreter will first scans the input file, parse the resulting tokens and creates an abstract syntax tree (AST). After that the compiler will check the AST and then output a cpp file. Similarly, the interpreter will also check the AST but print out the summary to screen after interpreting.

5.2 Scanner

The `EZMath` scanner tokenizes the input into `EZMath` readable units.

A typical LaTeX file contains two sections: text and math formula between $$. At the same time, LaTeX file has its own comment, which starts with %.

Our scanner will discard whitespace, LaTeX comment and text and only focus on the math formula section. Illegal character combinations, such as unsupported math operations are caught here. The scanner is based on ocamllex.

5.3 Parser

The `EZMath` parser generates AST from the tokens provided by the scanner. The parser matches the sequence of tokens with the rules defined in LRM. Mathed LaTeX math formulas will be constructed into AST. Otherwise, syntax errors will be caught here. The parser is based on ocamlyacc.

5.4 Abstract Syntax Tree

The abstract syntax tree is the intermediate representation of matched LaTeX math formulas after it has been parsed but before it has been semantically checked. There are two fundamental sections of AST: formula list and statement list. Formula is something like function in other programming language. Statement includes all other supported expressions.

5.5 Interpreter

The `EZMath` interpreter takes in the AST from parser and interpret it. The interpreting includes three sections:

1. Semantics Validation
2. Expression Evaluation
3. Formula Execution

The interpreter processes expression with OCaml build-in operations. Formula in AST will first be converted into expressions with local variables and then be evaluated. Before any evaluation, the interpreter will do semantic check to make sure everything works. Eventually, evaluated formulas and expressions will be displayed on screen as a summary of the input LaTeX file.

5.6 Compiler

The `EZMath` compiler takes in the AST from the parser and translate it into c++ file. The translation includes three sections:

1. Semantics Validation
2. Expression Translation
3. Formula Translation

The compiler translates expressions and formulas in AST into c++ style declarations, definitions and instructions (statements) by using tools written by our team in OCaml. Before any translation, the compiler will do semantic check. The compiler need to maintain all the variables and functions in the output c++ program, which will be fundamental for performing evaluation in the c++ program. And finally, the compiler will output a cpp file.

Chapter 6

Test Plan

6.1 Automated Testing

Adopting the manner from Microc test, shell test scripts `testall.sh` are used to run all the test cases `(*.tex)` automatically and compare the results with the expected results. For each test case, it reports OK if the result is expected, it reports FAILED otherwise.

Given the fact that there are several options `(-a, -i, -c)` when running the EZ-Math compiler, and different types of output and output files can be generated, test cases are divided into two groups.

The first group called SUCCESS is to test if all the specified features in LRM can be successfully achieved. The SUCCESS folder architecture is as follows:

```
tests
|-- SUCCESS
|   |-- src
|   |   |-- *.tex
|   |-- ref
|   |   |-- interpret
|   |   |   |-- *.out
|   |   |-- compile
|   |   |   |-- *.tex
|   |-- output
|   |   |-- interpret
|   |   |   |-- *.out
|   |   |   |-- *.diff
|   |   |-- compile
|   |   |   |-- *.tex
|   |   |   |-- *.diff
```

- `src` folder holds all the source files `(*.tex)`.

- `ref` folder holds all the expected outcome.

  - `interpret` folder holds all the expected outcome for compiling with -i option.

  - `compile` folder holds all the expected outcome for compiling, which are the final expected report files `(result.tex)`.

- `output` folder holds all the real output when running the script.

  - `interpret` folder holds all the real outcome for compiling with -i option and the diff file between the real and expected outcome.

  - `compile` folder holds all the real outcome for compiling, which are the final real report files `(result.tex)`, and the diff file between the real and expected report files.

If the test case success for both interpret and compiling option, it should also success for ast option. Additionally, the cpp file which generates the report file should also be successful.

To ensure all the exceptions are caught correctly, second group called FAIL are to test that the compiler can successfully catch and raise all the exceptions. The FAIL folder architecture is as follows:

```
tests
|-- FAIL
|   |-- src
|   |   |-- *.tex
|   |-- ref
|   |   |-- *.out
|   |-- output
|   |   |-- *.out
|   |   |-- *.diff
```

- `src` folder holds all the source files `(*.tex)`

- `ref` folder holds all the expected exception for compiling with -i option

- `output` folder holds all the real outcome for compiling with -i option and the diff file between the real and expected outcome.

All the detailed testing processes are additionally logged to the file `testall.log`.

The `testall.sh` can run with an option `-k` which will keep all the intermediate files generated.

6.2 Test Cases

There are two types of test cases, the ones that are expected to raise an exception are titled fail*.tex, the others that are expected not to raise any exception are titled test*.tex. Some special test cases that can demonstrate in-depth language features are listed below and explained in detail.

The example below shows that EZMath uses applicative evaluation order and parameters for formula are evaluated from right to left.

```
tests/SUCCESS/src/test-formula1.tex
x = 5
f(3,4)
f(x=f(x,x), x=f(x,x))
f(x,y) = y+1

output:
title{(No Title)} author{Unknown Author}
----------------------------------------
         Formular Definitions
----------------------------------------
f(x, y) = y+1
----------------------------------------
         Formular Evaluation
----------------------------------------
f(3,4) = 5
f(7,6) = 7
----------------------------------------
         Logical Validation
----------------------------------------
----------------------------------------
         Variable Definitions
----------------------------------------
x = 7
----------------------------------------
         Matrix Definitions
----------------------------------------
```

The example below shows that when calling a piecewise formula, EZMath will go through each condition from the beginning and return immediately when it finds the first matching condition.

```
tests/SUCCESS/src/test-piecewise1.tex
x = 7
f(a,b) =
\begin{cases}
3 & x>4 \\
1 & x>4
\end{cases}
f(1,1)
```

```
output:
title{(No Title)} author{Unknown Author}
----------------------------------------
          Formular Definitions
----------------------------------------
f(a, b) = {
    3, if x>4. Or
    1, if x>4.
}
----------------------------------------
          Formular Evaluation
----------------------------------------
f(1,1) = 3
----------------------------------------
          Logical Validation
----------------------------------------
----------------------------------------
          Variable Definitions
----------------------------------------
x = 7
----------------------------------------
          Matrix Definitions
----------------------------------------
```

Chapter 7

Lessons Learned

7.1 Voices of the Members

Piaoyang Cui I think the first lesson I learned from this project is how complicated it is for modern language compiler like C++ or Java. Even though our language is relative small, there are enormous details needed to be considered in the compiler, and at times different details can be entangled together which makes it surprisingly hard to do every thing right. This wins my respect to the mainstream compiler projects.

The second lesson is the importance of "Get things right" over "Get things done". For software projects like compiler, the popular metric of Line-of-Code is irrelevant. It can't reflect the productivity in any perspective. After all, writing more code quickly doesn't make sense if it fails to work properly or was bad designed. Code in compiler should be the result of deliberation.

Also I found the great power of efficient teamwork. Our teammates are all very dedicated people and we luckily established some principals to help we improve the productivity. In the end we were able to acheive things beyond the capacity of a personal project.

Yi Wang For me personally, I've learned a lot from various aspects, including technical improvements and teamwork collaboration ability. First of all, I participated in the whole process from scanner to parser, from abstract syntax construction to generate viable output, which makes me understand compiler work flow much better. Some concepts seem trivial to me before the course become much complicated, like the type checking process. Some concepts seem so hard for me to understand become much clearer, like the method to distinguish minus sign with negative number. Moreover, I've learned how to write functional programs, functional program language like OCaml is hard at first

sight, but easier when getting along in some aspects compared to traditional language, it's hard to imagine how complicated it is to write compiler in C++.

Through the project, I've learned the importance of participating a successful team with talented teammates. Communication is always the most important factor to make achievements in the team, and discussions even debates make every single decision wise and visionary. Tools are the basics for us to communicate and exchange ideas, good coding habits and software development habits would always minimize the costs of mistakes.

Shangjin Zhang As far as I'm concerned, teamwork is very important. I am really proud of our team. Everyone is diligent and trustworthy. Assigning each member a suitable role is very necessary for such a large project.

Also, tools can be really helpful when building a big project. We found several online editing and running websites, which make it possible for multi-tasking. Everyone can contribute at the same time. Version control helps avoid potential file-missing disaster. Online text editing increases the efficiency.

Moreover, testing is a non-negligible part. Thanks to the black-box testing, we can find bugs that the designer will always ignore. We put all detected bugs into a pool. Each member then picked one, fixed it, tested it, committed it and picked another one.

Finally, we need dictators in our team. Usually, there are so many possible solutions or directions for the project. Endless discussion can never produce any progress.

Zhejiao Chen The first lesson I have learned is that how important your teammates are when you are doing a semester long project. Working together with my skillful, experienced, professional and trustworthy teammates has not only motivated me to get involved in the project, but also helped me learn a great deal of things from development tools to logical thinking during the process. So it's never a bad idea to spend some time searching for your teammates before forming a group.

Secondly, it will save you a lot more effort later if you spend time learning how to use tools in the first place. Though it may be time consuming to read documents for the first time, it will speed up your work later and improve the whole efficiency and reliability of your project.

Thirdly, it's very important to spend more time thinking through the system goal and architecture before you actually start the work. Getting into coding directly without a delicately designed architecture and big view of the picture can lead to messy bugs and situations later which just can't be solved without starting from the beginning again.

7.2 Advice for Future Teams

For the language designing, we suggest future teams to focus on the direction and goal of your language, instead of debating on the details. Details can easily revised during the development but the goal usually remains the same. Let the Language Reference Manual (LRM) open to change, but should keep team members informed about all the decisions.

In addition, we suggest future teams to pay attention to the tools and development environments. How you communicate with each other in a daily manner and how you synchronized your work is of much importance and modern tools can help you reduce the overhead in these processes. This has been proved true during our development experience.

And, as always, start early and good luck!

Chapter 8

Appendix

8.1 Project Log

```
2014-12-15  Piaoyang_Cui  Merge branch 'tests' of https://
    github.com/i3wangyi/EZMath
2014-12-15  Piaoyang_Cui  Add comment in compile.ml
2014-12-15  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-15  Piaoyang_Cui  More points
2014-12-15  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-15  Shangjin_Zhang  add sum in interpreter
2014-12-15  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath into tests
2014-12-15  Yi_Wang Modify testall script
2014-12-15  Piaoyang_Cui  Add comment in parser
2014-12-15  Piaoyang_Cui  Continue adding comment
2014-12-15  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-15  Piaoyang_Cui  Add comment
2014-12-15  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-15  Yi_Wang add sectionial comments
2014-12-15  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-15  Shangjin_Zhang  implement sum as one type of
    expression
2014-12-15  Yi_Wang Add Makefile
2014-12-15  Piaoyang_Cui  Add negative sign
2014-12-15  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-15  Piaoyang_Cui  Add more points
```

```
2014-12-15  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-15  Yi_Wang change transpose in matrix, add pi, e
2014-12-15  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-15  Piaoyang_Cui  Remove non-use file
2014-12-15  Piaoyang_Cui  fix issue#20
2014-12-15  Yi_Wang separate header and compile code
    generation hard code
2014-12-15  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-15  Yi_Wang Add tan, log10, modify output cppfile
2014-12-15  Piaoyang_Cui  Fix help messgae
2014-12-15  Yi_Wang remove test files
2014-12-15  Yi_Wang Add ignore file, pass basic test
2014-12-15  Yi_Wang Merge branch 'master' into uniop
2014-12-15  Piaoyang_Cui  Fix issue no.19
2014-12-15  Yi_Wang modify interpret to print author title
2014-12-15  Yi_Wang modify test paper, add {} in every power
     function
2014-12-15  Piaoyang_Cui  Collecting ideas for slides
2014-12-15  Piaoyang_Cui  Change final _ case message
2014-12-14  Yi_Wang fix uniop in compile
2014-12-14  Piaoyang_Cui  Superstar toplevel
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath into uniop
2014-12-14  Yi_Wang exponent must be inside the BRACE pair
2014-12-14  Yi_Wang add sin cos log function in interpret
2014-12-14  Piaoyang_Cui  Update readme status
2014-12-14  Piaoyang_Cui  Improve format
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Improve interpreter output
2014-12-14  Zhejiao_Chen  add testall.sh
2014-12-14  Zhejiao_Chen  add testall.sh
2014-12-14  Piaoyang_Cui  Ref in interpreter
2014-12-14  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-14  Shangjin_Zhang  Seperate function declaration
    and definition
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang Apply float2str to all related files
2014-12-14  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-14  Shangjin_Zhang  change stringstream to dtos
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang Add float2str for truncated float num
2014-12-14  Zhejiao_Chen  Merge branch 'master' of https://
```

```
    github.com/i3wangyi/EZMath
2014-12-14  Zhejiao_Chen  remove temp.tex
2014-12-14  Piaoyang_Cui  Merge branch 'master' into
    full_interpreter
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Ignore result.tex
2014-12-14  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-14  Shangjin_Zhang  make logical expression look
    good
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Add double_to_string func
2014-12-14  Zhejiao_Chen  issue#8 fixed; add relevant test-
    arith4.tex
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang pretty print pdf through generated latex
2014-12-14  Zhejiao_Chen  issue#15 fixed; add test case fail
    -formula7.tex
2014-12-14  Zhejiao_Chen  issue#13 fixed; add two test cases
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Fix issue#15
2014-12-14  Zhejiao_Chen  add test case;issue#15: matrix in
    formula no exception
2014-12-14  Piaoyang_Cui  Remove token RPAREN_ASSIGN
2014-12-14  Zhejiao_Chen  issue#9 fixed; add relevant new
    test case to temp_test/test-formula3.tex
2014-12-14  Piaoyang_Cui  Merge branch 'master' into
    full_interpreter
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Continue fix issue no. 9
2014-12-14  Zhejiao_Chen  continuing issue#9, another test
    case that fails
2014-12-14  Piaoyang_Cui  Fix issue no.14
2014-12-14  Zhejiao_Chen  issue matrix +- fix change issue
    -1.tex to test-matrix6.tex
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang Fix var name issue in compile
2014-12-14  Piaoyang_Cui  Change StringMap to StringSet in
    function parameter unique test
2014-12-14  Yi_Wang Fix interpret, cannot assign a var with
    formular id
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Fix issue no.9 (boolean ==)
```

```
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Add matrix +/- in interpreter
2014-12-14  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Zhejiao_Chen  add issue-1.tex: matrix +/- not
    supported
2014-12-14  Piaoyang_Cui  == should be higher than =. fix
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  add stdexcept
2014-12-14  Yi_Wang update
2014-12-14  Piaoyang_Cui  Add platform test goal
2014-12-14  Yi_Wang Print 1st version latex by C++
2014-12-14  Yi_Wang Last step to fix the C++
2014-12-14  Piaoyang_Cui  Add overloading to compiler
2014-12-14  Piaoyang_Cui  Fix function redefinition
2014-12-14  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-14  Yi_Wang reset scanner
2014-12-14  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-14  Piaoyang_Cui  Try to solve the overloading
2014-12-13  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Zhejiao_Chen  add test cases
2014-12-13  Yi_Wang fix negative number issue
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang pretty print with formular definition in
     latex
2014-12-13  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Zhejiao_Chen  add test cases
2014-12-13  Piaoyang_Cui  Unify all the raise_Failure
    message (Upcase, space, etc)
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang pretty print cpp file
2014-12-13  Piaoyang_Cui  Fix issue of parameter name not
    unique
2014-12-13  Piaoyang_Cui  add parameter test
2014-12-13  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Zhejiao_Chen  add test cases
2014-12-13  Piaoyang_Cui  Update matrix exception message
2014-12-13  Piaoyang_Cui  Delete return -1.0
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
```

```
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang Modify cpp print order
2014-12-13  Piaoyang_Cui  Handle exception in piecewise
    function
2014-12-13  Piaoyang_Cui  Change error processing
2014-12-13  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Piaoyang_Cui  Add header file for std::
    runtime_error
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang Fix Preamble and modify deprecated
    methods
2014-12-13  Piaoyang_Cui  Fix run.sh
2014-12-13  Piaoyang_Cui  Fix Not_found error
2014-12-13  Shangjin_Zhang  reverse some lists in compile
2014-12-13  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-13  Shangjin_Zhang  merge full_compile manually
2014-12-13  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Piaoyang_Cui  Remove warning
2014-12-13  Piaoyang_Cui  Add function call literal
2014-12-13  Yi_Wang Modify ignore file
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang add author and title in top level
2014-12-13  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Zhejiao_Chen  add test cases
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang add regular expression for title and
    author
2014-12-13  Piaoyang_Cui  Full functional interpreter
2014-12-13  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i2wangyi/EZMath
2014-12-13  Zhejiao_Chen  add temp_test
2014-12-13  Piaoyang_Cui  fix issue no.2 and test
2014-12-13  Piaoyang_Cui  Resume run.sh
2014-12-13  Piaoyang_Cui  fix issue no.1
2014-12-13  Piaoyang_Cui  Add Logo in version
2014-12-13  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-13  Yi_Wang combine header's hard code with
    generated sets
2014-12-13  Piaoyang_Cui  Delete a.out
2014-12-13  Piaoyang_Cui  Modify .gitignore
2014-12-13  Piaoyang_Cui  Delete unuseful files
2014-12-13  Piaoyang_Cui  Merge branch 'master' of https://
```

```
    github.com/i3wangyi/EZMath
2014-12-13  Piaoyang_Cui  Improve toplevel: (1) Add version,
     help info and asscoative options (2) add argv validation
     (missing, too many, not valid) (3) compatible with
    different input
2014-12-13  Yi_Wang modify header output, start merge header
     with i_list
2014-12-13  Yi_Wang add version info
2014-12-13  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-13  Piaoyang_Cui  Update readme.md
2014-12-11  Zhejiao_Chen  add all raise failure tests for
    interpret.ml
2014-12-11  Zhejiao_Chen  Add test cases, fix bug with
    comment inside code entry
2014-12-10  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-10  Zhejiao_Chen  modify test result
2014-12-10  Yi_Wang Add test output
2014-12-10  Yi_Wang fix run fail test
2014-12-10  Yi_Wang Add check Fail in automated test
2014-12-10  Yi_Wang redirect stderr to stdin
2014-12-10  Yi_Wang Add test for interpret result
2014-12-10  Yi_Wang add test log function
2014-12-10  Yi_Wang testall script, success test case
2014-12-09  Yi_Wang Matrix Literal check at parser time
2014-12-09  Shangjin_Zhang  integrate matrix_to_string_c
    into expr_to_string_c
2014-12-09  Piaoyang_Cui  Matrix_(row)_to_string_C
2014-12-09  Shangjin_Zhang  using stringstream now
2014-12-09  Shangjin_Zhang  Compile draft (can test with run
    .sh)
2014-12-09  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-09  Shangjin_Zhang  unname matrix
2014-12-09  Piaoyang_Cui  Allow anonymous list to contruct a
     matrix
2014-12-09  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-09  Yi_Wang Remove sample.cpp
2014-12-09  Yi_Wang remove sample.cpp in gitignore
2014-12-09  Shangjin_Zhang  merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-12-09  Shangjin_Zhang  Before merged from piaoyang
2014-12-09  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-12-09  Yi_Wang Test on Windows
2014-12-09  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-12-09  Piaoyang_Cui  Sync with header.ml
```

```
2014-12-08  Yi_Wang Merge branch 'master' of https://github.
   com/i3wangyi/EZMath
2014-12-08  Piaoyang_Cui  Update matrix class
2014-12-08  Yi_Wang Add testall.sh from microC
2014-12-08  Piaoyang_Cui  Interpret -- top level function
2014-12-08  Shangjin_Zhang  merge expr.ml to header.ml
2014-12-08  Yi_Wang change sequence of print strings
2014-12-08  Yi_Wang Modify hard code print, in header.ml
2014-12-07  Yi_Wang modify test_file.sh, present printed cpp
    in stdout
2014-12-07  Yi_Wang Get system time implemented, write to
   file operation implemented, made in separate ml file
2014-12-06  Shangjin_Zhang  Now we have all tools for
   reversing ast to latex
2014-12-06  Piaoyang_Cui  Merge branch 'master' of https://
   github.com/i3wangyi/EZMath
2014-12-06  Piaoyang_Cui  Add matrix type
2014-12-05  Shangjin_Zhang  expr_to_string
2014-12-05  Piaoyang_Cui  Copy interpret.ml to compile.ml
2014-12-05  Shangjin_Zhang  Change space to any space in
   scanner.mll
2014-12-05  Shangjin_Zhang  Change space to any space in
   scanner.mll
2014-12-05  Piaoyang_Cui  Eliminate semi
2014-12-05  Shangjin_Zhang  add compile.ml
2014-12-05  Yi_Wang mv compile to interpret
2014-12-05  Yi_Wang delete output file
2014-12-05  Yi_Wang Merge branch 'redesign' of https://
   github.com/i3wangyi/EZMath into redesign
2014-12-05  Yi_Wang Update
2014-12-05  Shangjin_Zhang  change order of float*matrix
2014-12-05  Yi_Wang Add matrix print
2014-12-05  Yi_Wang Update
2014-12-05  Yi_Wang Modify the content, parser error
2014-12-05  Yi_Wang Update README, TODO, delete temp files
2014-12-05  Shangjin_Zhang  Fix \% comment in latex
2014-12-05  Yi_Wang modify sys.argv operations
2014-12-04  Yi_Wang Merge branch 'matrix_operations2' of
   https://github.com/i3wangyi/EZMath into
   matrix_operations2
2014-12-04  Yi_Wang add constant matrix multiplication
2014-12-04  Shangjin_Zhang  Make matrix looks better
2014-12-04  Shangjin_Zhang  Add dotMul and transpose; Tested
2014-12-04  Yi_Wang modify matrix multiplication method
2014-12-04  Yi_Wang Add matrix multiplication
2014-12-04  Yi_Wang Convert matrix definition from list to
   float array
2014-12-04  Shangjin_Zhang  remove one warning
2014-12-04  Shangjin_Zhang  Can print matrix in both -c and
   -a
```

```
2014-12-03  Piaoyang_Cui  Add parser.txt exmaples
2014-12-03  Piaoyang_Cui  Complete matrix
2014-12-03  Piaoyang_Cui  Add scanner for times
2014-12-01  Piaoyang_Cui  Cool
2014-12-01  Piaoyang_Cui  fix bugs
2014-12-01  Piaoyang_Cui  Try TopLevel
2014-12-01  Piaoyang_Cui  1st version of compile.ml
2014-12-01  Piaoyang_Cui  sectionial Complete Compile.ml,
    without test
2014-12-01  Piaoyang_Cui  Continue update compile.ml
2014-12-01  Piaoyang_Cui  Update compile.ml -- interpret
2014-12-01  Piaoyang_Cui  Copy from MicroC as comment
2014-12-01  Yi_Wang Modify Read File Function
2014-11-26  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-26  Zhejiao_Chen  add test1.tex
2014-11-26  Piaoyang_Cui  Change EZMath.ml
2014-11-26  Piaoyang_Cui  Create compile.ml
2014-11-26  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-26  Yi_Wang add "\$\$" in front of tex file
2014-11-26  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-11-26  Shangjin_Zhang  Create top level file
2014-11-26  Piaoyang_Cui  Remove useless calc.ml
2014-11-26  Piaoyang_Cui  Clean the repo
2014-11-26  Piaoyang_Cui  Fix FID with (
2014-11-26  Piaoyang_Cui  Fix matrix
2014-11-26  Shangjin_Zhang  Reverse Seq
2014-11-26  Shangjin_Zhang  Reverse displayed list
2014-11-26  Shangjin_Zhang  small fix with ast.ml
2014-11-26  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-11-26  Shangjin_Zhang  Print AST
2014-11-26  Piaoyang_Cui  Update README
2014-11-26  Piaoyang_Cui  Change LRM: add statement section,
     usage of comma, semi, prohibted IDs
2014-11-26  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-26  Piaoyang_Cui  Add regression test of scanner
2014-11-26  Zhejiao_Chen  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-26  Jessie Chen add tests folder
2014-11-26  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-26  Yi_Wang sectionial Evaluate syntax tree
2014-11-26  Piaoyang_Cui  Merge branch 'scanner_test'
2014-11-26  Piaoyang_Cui  Move to a new folder
2014-11-26  Piaoyang_Cui  Create a regression test of
    Scanner
```

```
2014-11-26  Yi_Wang Merge branch 'parser_fix' of https://
    github.com/i3wangyi/EZMath into parser_fix
2014-11-26  Yi_Wang add evaluate function in ast
2014-11-26  Piaoyang_Cui  Merge branch 'parser_fix' of https
    ://github.com/i3wangyi/EZMath into parser_fix
2014-11-26  Piaoyang_Cui  Add to-do of LRM to README
2014-11-26  Yi_Wang Remove pdf
2014-11-26  Yi_Wang Update the TO-DO and Progress in the
    ReadME
2014-11-26  Yi_Wang Merge branch 'parse_test' of https://
    github.com/i3wangyi/EZMath into parse_test
2014-11-26  Shangjin_Zhang  Add 34642 to paper.tex; Add Pow(
    CARET); Change parser entry in parser_test.ml
2014-11-25  Piaoyang_Cui  Merge branch 'parser_fix' of https
    ://github.com/i3wangyi/EZMath into parser_fix
2014-11-25  Piaoyang_Cui  Now scanner should start with (and
     if out previous, will return to) code mode, and the
    input file should add two \$\$ in the beginning`
2014-11-25  Piaoyang_Cui  An example of new simple.tex: add
    two \$\$ in the beginning of file
2014-11-25  Piaoyang_Cui  Fix: code mode should not have EOF
2014-11-25  Piaoyang_Cui  Fix the fdecl: FID has included
    LPAREN
2014-11-25  Shangjin_Zhang  new branch for test
2014-11-24  Piaoyang_Cui  Resolve all the conflicts
2014-11-24  Piaoyang_Cui  Resolve the last reduce/reduce
    conflict (1 S/R left)
2014-11-24  Piaoyang_Cui  Try to resolve exec_expr and expr
2014-11-24  Piaoyang_Cui  Reduce 2 conflicts
2014-11-24  Piaoyang_Cui  Reduce 1st conflict
2014-11-24  Yi_Wang target conflict at statement
2014-11-24  Yi_Wang Update Parser.mly without statement
2014-11-24  Yi_Wang Parser fix
2014-11-24  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-24  Yi_Wang Local Modification
2014-11-24  Piaoyang_Cui  Change Block to Seq
2014-11-24  Yi_Wang modify gitignore
2014-11-24  Yi_Wang Delete intermediate files
2014-11-24  Yi_Wang add gitignore
2014-11-24  Yi_Wang open a new line
2014-11-24  Yi_Wang update the variable number
2014-11-24  Yi_Wang modify definition of Matrix in Parser
2014-11-23  Yi_Wang Add to-do, resolve ast.mli symbol error
2014-11-19  Piaoyang_Cui  Fix the parser.mly
2014-11-19  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-11-19  Shangjin_Zhang  Delete one reference matrix
2014-11-19  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
```

```
2014-11-19  Yi_Wang Modify parser
2014-11-19  Shangjin_Zhang  Modified Parser.mly
2014-11-19  Shangjin_Zhang  Add Matrix to Ast.mli
2014-11-19  Shangjin_Zhang  Modified Ast.mli
2014-11-19  Yi_Wang add real token in parser, modify matrix
    definition in scanner
2014-11-19  Yi_Wang add naive matrix scanner, separate int
    and real number
2014-11-19  Yi_Wang remove unused test file
2014-11-19  Yi_Wang add integer scanner
2014-11-19  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-19  Yi_Wang update integer to float number in the
    scanner
2014-11-19  Piaoyang_Cui  Change regression test of scanner
    -- according to new paper.tex
2014-11-19  Shangjin_Zhang  Small change in paper.tex (= ->
    ==)
2014-11-19  Piaoyang_Cui  Add a naive regression test of
    scanner
2014-11-19  Shangjin_Zhang  Test scanner (Please run compile
    .sh)
2014-11-18  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-11-18  Shangjin_Zhang  Add 3 more files (Please ignore
    the content inside)
2014-11-17  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-17  Piaoyang_Cui  Change rule names of scanner.mll
2014-11-16  Shangjin_Zhang  Add Tokens in Scanner
2014-11-16  Shangjin_Zhang  Merge branch 'master' of https
    ://github.com/i3wangyi/EZMath
2014-11-16  Shangjin_Zhang  Add scanner.mll
2014-11-16  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-16  Piaoyang_Cui  Tesing conflict resolving 2
2014-11-16  Yi_Wang Testing conflict resolving
2014-11-16  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-16  Piaoyang_Cui  Merge branch 'master' of https://
    github.com/i3wangyi/EZMath
2014-11-16  Piaoyang_Cui  Testing conflict resolve
2014-11-16  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
2014-11-16  Yi_Wang Update ReadMe
2014-11-16  Shangjin Zhang  Add scanner.mll
2014-11-16  Shangjin Zhang  Test Push Directly
2014-11-16  Piaoyang Cui  Added course info
2014-11-16  Yi_Wang Merge branch 'master' of https://github.
    com/i3wangyi/EZMath
```

```
2014-11-16  Yi_Wang add ignore
2014-11-16  Yi Wang Merge pull request #1 from
    Shangjin_Zhang/master
2014-11-16  Shangjin Zhang  Test Github
2014-11-16  Yi_Wang Update ReadMe, test the slack
    integration
2014-10-27  Yi_Wang Update LRM
2014-10-27  Yi_Wang Add Sample C++ file
2014-10-27  Yi Wang Initial commit
\end
```

8.2 Source Code

This appendix contains the source code for EZMath. Counts are not included for test cases or demos.

File Name	Lines of Code
EZMath.ml	82
header.ml	411
interpret.ml	378
compile.ml	564
scanner.mll	72
parser.mly	143
ast.ml	98
Makefile	45
testall.sh	121

Listing 8.1: ast.ml

```
type op = Add | Sub | Mul | Div | Equal | Neq | Less | Leq |
    Greater | Geq | Pow | DotMul
type uop = Sin | Cos | Log | Trans | Tan

type matrix = float array array

type expr = (* Expressions *)
Literal of float (* e.g. 42 *)
|   MLiteral of matrix
|   Id of string (* e.g. foo *)
|   Assign of string * expr (* e.g. foo = 42 *)
|   Binop of expr * op * expr (* e.g. a + b *)
|   Call of string * expr list (* e.g. foo(1, 25) *)
|   Uniop of uop * expr (* e.g. a^{T} *)
|   Sum of expr * expr * expr (* e.g. /sum_{i=1}^{10}{i} *)
|   Prod of expr * expr * expr (* e.g. /prod_{i=1}^{10}{i}
    *)

type statement = (* Statements *)
Seq of expr list (* e.g. x=3, y=4, z=5 *)

type case = {
expression : expr;
condition : expr;
}

type f_def = (* Formular Definition *)
Regular of expr (* e.g. b(m,n) = 2*n*m *)
|   Piecewise of case list (* e.g. a(m,n) = \begin{cases} \
    end{cases}*)

type formular = {
fname : string;
parameter : expr list;
definition : f_def;
}

type program = formular list * statement list

(* AST printing utilities. Use '-a' in top-level to enable
    *)

(* Return a truncated string of  *)
let float2str f =
let i = int_of_float f in
if float_of_int i = f then string_of_int i
else string_of_float f
;;
```

```
45
46 (* Matrix printing utilities *)
47 let matrix_s m =
48 " [" ^ String.concat ", " (Array.to_list (Array.map (fun e
       -> "(" ^ (float2str e) ^ ")") m)) ^ " ]"
49
50 let traverse_matrix any = Array.to_list (Array.map (fun e ->
       "(" ^ matrix_s e ^ ")") any)
51
52 let matrix_decl_s m =
53 "{\n         " ^ String.concat ",\n         " (
       traverse_matrix m) ^ "]\n}"
54
55 (* Expression/statement printing utilities *)
56 let rec expr_s = function
57 Literal(l) -> "Literal " ^ float2str l
58 | MLiteral(m) -> "MLiteral " ^ matrix_decl_s m
59 | Id(s) -> "Id " ^ s
60 | Binop(e1, o, e2) -> "Binop (" ^ expr_s e1 ^ ") " ^
61 (match o with Add -> "Add" | Sub -> "Sub" | Mul -> "Mul" |
62 Div -> "Div" | Equal -> "Equal" | Neq -> "Neq" |
63 Less -> "Less" | Leq -> "Leq" | Greater -> "Greater" |
64 Geq -> "Geq" | Pow -> "Pow" | DotMul -> "DotMul") ^ " (" ^
       expr_s e2 ^ ")"
65 | Sum (down, up, e) -> "Sum (" ^ (expr_s down) ^ ")" ^ "(" ^
       (expr_s up) ^ ")" ^ "(" ^ (expr_s e) ^ ")"
66 | Prod (down, up, e) -> "Prod (" ^ (expr_s down) ^ ")" ^ "("
       ^ (expr_s up) ^ ")" ^ "(" ^ (expr_s e) ^ ")"
67 | Uniop(o, e) -> "Uniop (" ^
68 (match o with Sin -> "Sin" | Trans -> "Trans" | Cos -> "Cos"
       | Log -> "Log"
69 | Tan -> "Tan"
70 ) ^ " (" ^ expr_s e ^ ")"
71 | Assign(v, e) -> "Assign " ^ v ^ " (" ^ expr_s e ^ ")"
72 | Call(f, es) -> "Call " ^ f ^ " [" ^
73 String.concat ", " (List.map (fun e -> "(" ^ expr_s e ^ ")")
       es) ^ "]"
74
75 let traverse any = List.map (fun e -> "(" ^ expr_s e ^ ")")
       any
76
77 let stmt_s = function
78 Seq(ss) -> "Seq[" ^ String.concat ",\n"
79 (traverse ss) ^ "]"
80 (* Function priting utilities *)
81 let case_s c =
82 "\n { expression = \"" ^ (expr_s c.expression) ^ "\"\n
       condition = [" ^
83 (expr_s c.condition) ^ "] }\n"
84
```

```
let traverse_case any = List.map (fun e -> "(" ^ case_s e ^
    ")") any

let f_def_s = function
Regular(r) -> "Regular (" ^ expr_s r ^ ")"
| Piecewise(p) -> "Piecewise[" ^ String.concat ",\n"
(traverse_case p) ^ "]"

let func_decl_s f =
" { fname = \"" ^ f.fname ^ "\"\n   parameter = [" ^
String.concat ", " (traverse f.parameter) ^ "]\n definition
    = [\n"
^ (f_def_s f.definition) ^ " ]}\n"

(* Overall program printing *)
let program_s (funcs, stats) = "([" ^ String.concat "\n" (
    List.map func_decl_s funcs) ^ "],\n[ " ^ String.concat "\
    n" (List.map stmt_s stats) ^ "])\n"
```

Listing 8.2: scanner.mll

```
{
  open Parser
}

let digit = ['0'-'9']
let integer = digit+
let fraction = digit+
let exponent = ['e''E']['+''-']?digit+
let ident = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9']*

(* EZMath code. Scanner will start from here *)
rule code = parse
    [' ' '\t' '\r' '\n']  { code lexbuf }
  |   "$$"          { text lexbuf }
  |   '%'           { comment_in_code lexbuf}
  |   'T'           { TRANS }
  |   "\\pi"          { FLOAT(3.1415926) }
  | 'e'           { FLOAT(2.71828)}
  |   (ident as fid) [' ' '\t' '\r' '\n']*'('  { FID(fid) }
  | '_'           { UNDERSCORE }
  | '('           { LPAREN }
  | ')'           { RPAREN }
  | '{'           { LBRACE }
  | '}'           { RBRACE }
  | ','           { COMMA }
  | '+'           { PLUS }
  | '-'           { MINUS }
  | '*' | "\\times"     { TIMES }
  | '/'           { DIVIDE }
```

```
30 |   | "=="          { EQUAL }
31 |   | '='           { ASSIGN }
32 |   | "!="          { NEQ }
33 |   | ">="          { GEQ }
34 |   | "<="          { LEQ }
35 |   | '>'           { GT }
36 |   | '<'           { LT }
37 |   | "\\sum"         { SUM }
38 |   | "\\prod"        { PROD }
39 |   | "\\sin"         { SIN }
40 |   | "\\cos"         { COS }
41 |   | "\\log"         { LOG }
42 |   |   "\\tan"         { TAN }
43 |   | "\\begin"       { BEGIN }
44 |   | "\\end"         { END }
45 |   |   "\\cdot"        { DOTMUL }
46 |   | "cases"         { CASES }
47 |   | "bmatrix"       { BMATRIX }
48 |   | '&'           { AND }
49 |   | '^'           { CARET }
50 |   | "\\\\"          { DBACKSLASH }
51 |   | (integer? '.' fraction exponent | integer '.'? exponent
       |     | integer '.' fraction? | '.' fraction | integer) as
       |     num { FLOAT(float_of_string num)} (* Take integer as
       |     float number *)
52 |   | ident as vid { VID(vid) }
53 |   | _ as unknown { raise (Failure("illegal character " ^
       |     Char.escaped unknown)) }
54 |
55 | (* Normal Latex text *)
56 | and text = parse
57 |     '%'           { comment_in_text lexbuf }
58 |   | "$$"          { code lexbuf }
59 |   | eof           { EOF }
60 |   | _           { text lexbuf }
61 |
62 | (* Comment (%) in Latex text *)
63 | and comment_in_text = parse
64 |     '\n'          { text lexbuf }
65 |   | eof           { EOF }
66 |   | _           { comment_in_text lexbuf }
67 |
68 | (* Comment (%) in EZMath code *)
69 | and comment_in_code = parse
70 |     '\n'          { code lexbuf }
71 |   | eof           { raise (Failure("Missing end symbol $$"))
       |     }
72 |   | _           { comment_in_code lexbuf }
```

Listing 8.3: parser.mly

```
%{
open Ast
(* Convert matrix in 2d list into 2d array *)
let list2matrix data =
let m = List.length data in
if m <= 0 then raise (Failure ("Parse with matrix row == 0")
    ) else
let n = List.length (List.hd data) in
if n <= 0 then raise (Failure ("Parse with matrix col == 0")
    ) else
Array.of_list(
List.rev ((List.fold_left (fun l row -> (
if ((List.length row) <> n) then raise (Failure ("row item
    not consistent"))
else Array.of_list row)::l) [] data))
)
%}


/* Symbols */
%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA BEGIN END
    CASES BMATRIX CARET DBACKSLASH AND EOF
/* Float Operation */
%token ASSIGN EQUAL NEQ GEQ LEQ GT LT PLUS MINUS TIMES
    DIVIDE
%token SUM PROD UNDERSCORE
%token SIN COS LOG TAN
/* Matrix Binop */
%token DOTMUL TRANS
%token <float> FLOAT
%token <string> VID /* Variable ID both for float variable
    and matrix */
%token <string> FID /*Formular ID */

/*Precedence and Association */

%right ASSIGN
%left EQUAL NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE DOTMUL
%left CARET TRANS


%start program
%type <Ast.program> program

%%

program:
```

```
/* nothing */     { [], [] }
|   program fdecl     { ($2 :: fst $1), snd $1  } /* Prepend
    function decl */
|   program stmt      {  fst $1, ($2 :: snd $1) } /* Prepend
    statement */

/* For Matrix Data */
row_list :
row  { [List.rev $1] }
|   row_list DBACKSLASH row { (List.rev $3) :: $1 }

row :
FLOAT   { [$1] }
|   MINUS FLOAT { [-. $2] }
|   row AND FLOAT { $3 :: $1 }
|   row AND MINUS FLOAT { (-. $4)::$1 }

fdecl:
FID arguments RPAREN ASSIGN forumular_def
{
{
fname = $1;
parameter = $2;
definition = $5;
}
}
/* Regular function: single expr. Piecewise: match cases
   with exprs */
forumular_def :
regular_def     { $1 }
|   piecewise_def   { $1 }

regular_def :
expr                { Regular($1) }

piecewise_def :
BEGIN LBRACE CASES RBRACE case_list END LBRACE CASES RBRACE
   { Piecewise(List.rev $5) }

case_list :
case                        { [$1] }
|   case_list DBACKSLASH case { $3 :: $1 }

case :
expr AND expr
{
{
expression = $1;
condition = $3;
}
```

```
}

/* Top-level expression */
expr :
/* float constant*/
FLOAT                          { Literal($1) }
/* matrix constant */
|   BEGIN LBRACE BMATRIX RBRACE row_list END LBRACE BMATRIX
    RBRACE
{ MLiteral(list2matrix(List.rev $5)) }
|   VID                            { Id($1) }
|   exec_expr                      { $1 }

/* Expression allowed tp appear alone in statement */
exec_expr:
expr PLUS expr                 { Binop($1, Add, $3) }
|   expr MINUS expr                { Binop($1, Sub, $3) }
|   expr TIMES expr                { Binop($1, Mul, $3) }
|   expr DIVIDE expr               { Binop($1, Div, $3) }
|   expr EQUAL expr                { Binop($1, Equal, $3) }
|   expr NEQ expr                  { Binop($1, Neq, $3) }
|   expr LT expr                   { Binop($1, Less, $3) }
|   expr LEQ expr                  { Binop($1, Leq, $3) }
|   expr GT expr                   { Binop($1, Greater, $3)
    }
|   expr GEQ expr                  { Binop($1, Geq, $3) }
|   expr CARET LBRACE expr RBRACE  { Binop($1, Pow, $4) }
|   SUM UNDERSCORE LBRACE expr RBRACE CARET LBRACE expr
    RBRACE LBRACE expr RBRACE { Sum($4, $8, $11) }
|   PROD UNDERSCORE LBRACE expr RBRACE CARET LBRACE expr
    RBRACE LBRACE expr RBRACE { Prod($4, $8, $11) }
|   VID ASSIGN expr                { Assign($1, $3) }
|   FID arguments RPAREN          { Call($1, $2) }
|   LPAREN expr RPAREN             { $2 }
|   LPAREN MINUS expr RPAREN       { Binop(Literal(0.), Sub
    , $3)}
|   expr DOTMUL expr               { Binop($1, DotMul, $3)
    }
|   expr CARET LBRACE TRANS RBRACE  { Uniop(Trans, $1)  }
|   SIN LBRACE expr RBRACE          { Uniop(Sin, $3)    }
|   COS LBRACE expr RBRACE          { Uniop(Cos, $3)    }
|   LOG LBRACE expr RBRACE          { Uniop(Log, $3)    }
|   TAN LBRACE expr RBRACE          { Uniop(Tan, $3)    }

arguments:
{ [] }
|   argument_list                   { List.rev $1 }

argument_list :
expr                           { [$1] }
```

```
|   argument_list COMMA expr         { $3 :: $1 }

/* Only exec_expr allowed alone in statement */
stmt:
exec_expr_list                     { Seq(List.rev $1) }

exec_expr_list :
exec_expr                       { [$1] }
|   exec_expr_list COMMA exec_expr { $3 :: $1 }
```

Listing 8.4: header.ml

```
open Ast
open Unix (* Header for get system date *)

let version = "version " ^ "0.1";;

let date =
let months = [| "January"; "February"; "March"; "April"; "
    May"; "June"; "July"; "August"; "September"; "October"; "
    November"; "December" |]
in
let local_t = Unix.localtime (Unix.gettimeofday ()) in
months.(local_t.tm_mon) ^ ", " ^ string_of_int (local_t.
    tm_year + 1900)
;;

(* Generate header code in c++ *)
let header v_list m_list l_count c_count fdef_latex_l title
    author date =
"/**
*  EZMath
*  Fast documentation and computation of math-related text
*  Created by EZMath Compiler " ^ version ^ "
**/
"
^
"
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <cmath>
#include <stdexcept>

using namespace std;
"
^
"#define NUM_OF_VARIABLES " ^ string_of_int (List.length
    v_list) ^ "\n" ^
```

```
34 |
35 | "#define NUM_OF_MATRIX_VARIABLES " ^ string_of_int (List.
   |    length m_list) ^ "\n" ^
36 |
37 | "#define NUM_OF_LOGICAL_VALIDATION " ^ string_of_int l_count
   |     ^ "\n" ^
38 |
39 | "#define NUM_OF_FORMULAR_EVALUATION " ^ string_of_int
   |    c_count ^ "\n" ^
40 |
41 | "#define NUM_OF_FORMULAR_DEFINITION " ^ string_of_int(List.
   |    length fdef_latex_l) ^ "\n\n" ^
42 |
43 | "const string Title = \"" ^ "\\\\" ^ title ^ "\";\n" ^
44 |
45 | "const string Author = \"" ^ "\\\\" ^ author ^ "\";\n" ^
46 |
47 | "const string Date = \"" ^ date ^ "\";\n"
48 | ;;
49 |
50 | let matrix_class =
51 | "
52 | class matrix
53 | {
54 | //declare a vector of vectors of type double
55 | vector< vector<double> > s ;
56 | public:
57 | int m, n;
58 | //Initialize the size of s to row by col
59 | matrix(int row = 1, int col = 1): m(row), n(col), s(row,
   |    vector<double>(col)) {}
60 | matrix(int row, int col, const double data[]): m(row), n(col
   |    ), s(row, vector<double>(col))
61 | {
62 | for(int i = 0; i < row; i++)
63 | for(int j = 0; j < col; j++)
64 | s[i][j] = data[i * col + j];
65 | }
66 | string printm();
67 | //declare the operators +,-,*,~,DotMul as friends and with
   |    return type matrix
68 | friend matrix operator+(const matrix&, const matrix&);
69 | friend matrix operator-(const matrix&, const matrix&);
70 | friend matrix operator*(const matrix&, const matrix&);
71 | friend matrix operator*(const matrix&, double);
72 | friend matrix operator*(double, const matrix&);
73 | friend matrix operator~(const matrix&);
74 | friend matrix DotMul(const matrix&, const matrix&);
75 | };
76 | string matrix::printm()
```

```
77 {
78 stringstream lex;
79 lex << \"\\\\begin {bmatrix}\" << endl;
80 for(int i = 0; i < m; i++)
81 {
82 for(int j = 0; j < n; j++)
83 {
84 lex << this->s[i][j];
85 lex << ((j < n - 1) ? \" & \" : ((i < m - 1)? \" \\\\\\\\\" : \"\"));
86 }
87 lex << endl;
88 }
89 lex << \"\\\\end {bmatrix}\";
90 return lex.str();
91 }
92
93 matrix operator+(const matrix& a, const matrix& b)
94 {
95 //declare a matrix temp to store the result and return this matrix
96 if(a.m != b.m || a.n != b.n)
97 throw std::runtime_error(\"Invalid matrix addition: dimensions not match\");
98 matrix temp(a.m, a.n);
99 for(int i = 0; i < a.m; i++)
100 for(int j = 0; j < a.n; j++)
101 temp.s[i][j] = a.s[i][j] + b.s[i][j];
102 return temp;
103 }
104 matrix operator-(const matrix& a, const matrix& b)
105 {
106 if(a.m != b.m || a.n != b.n)
107 throw std::runtime_error(\"Invalid matrix subtraction: dimensions not match\");
108 matrix temp(a.m, a.n);
109 for(int i = 0; i < a.m; i++)
110 for(int j = 0; j < a.n; j++)
111 temp.s[i][j] = a.s[i][j] - b.s[i][j];
112 return temp;
113 }
114 matrix operator*(const matrix& a, const matrix& b)
115 {
116 if(a.n != b.m)
117 throw std::runtime_error(\"Invalid matrix multiplication: dimensions not match\");
118 matrix temp(a.m, b.n);
119 for(int i = 0; i < a.m; i++)
120 {
121 for(int j = 0; j < b.n; j++)
```

```
{
temp.s[i][j] = 0;
for(int k = 0; k < a.n; k++)
temp.s[i][j] += a.s[i][k] * b.s[k][j];
}
}
return temp;
}
matrix operator*(const matrix& a, double b)
{
matrix temp(a.m, a.n);
for(int i = 0; i < a.m; i++)
for(int j = 0; j < a.n; j++)
temp.s[i][j] = a.s[i][j] * b;
return temp;
}
matrix operator*(double b, const matrix& a)
{
matrix temp(a.m, a.n);
for(int i = 0; i < a.m; i++)
for(int j = 0; j < a.n; j++)
temp.s[i][j] = a.s[i][j] * b;
return temp;
}
matrix operator~(const matrix& trans)
{
matrix temp(trans.n, trans.m);
for(int i = 0; i < trans.m; i++)
for(int j = 0; j < trans.n; j++)
temp.s[j][i] = trans.s[i][j];
return temp;
}
matrix DotMul(const matrix& a, const matrix& b)
{
//declare a matrix temp to store the result and return this
    matrix
if(a.m != b.m || a.n != b.n)
throw std::runtime_error(\"Invalid matrix dot multiplication
    : dimensions not match\");
matrix temp(a.m, a.n);
for(int i = 0; i < a.m; i++)
for(int j = 0; j < a.n; j++)
temp.s[i][j] = a.s[i][j] * b.s[i][j];
return temp;
}
"
;;

let dtos =
"string dtos(double d)
```

```
{
stringstream out;
out << d;
return out.str();
}
"
;;

let main_preamble v_list m_list fdef_latex_l=
"
/**
* Main Function
**/
int main(int argc, char ** argv) {
"
^
"\tstring var_def[NUM_OF_VARIABLES] = {" ^ String.concat ",
    " (List.map (fun var -> "\"" ^ var ^ "\"") v_list) ^ "};\
    n" ^

"\tstring matrix_def[NUM_OF_MATRIX_VARIABLES] = {" ^ String.
    concat ", " (List.map (fun var -> "\"" ^ var ^ "\"")
    m_list) ^ "};\n\n" ^

"\tstring formular_def[NUM_OF_FORMULAR_DEFINITION] = {" ^
    String.concat ", \n" (List.map (fun f -> "\"" ^  f ^ "\""
     ) fdef_latex_l) ^ "};\n\n" ^

"\tstring l_result[NUM_OF_LOGICAL_VALIDATION];\n" ^

"\tstring c_result[NUM_OF_FORMULAR_EVALUATION];\n"
;;

let latex_print file =
"
/**
Latex Print Purpose
*/
ofstream file(\"" ^ file ^ "\");

//Begin
file << \"\\\\documentclass{article}\\n\"
<< \"\\\\usepackage[utf8]{inputenc}\\n\"
<< \"\\\\usepackage{amsmath}\\n\"
<<  Title << \"\\n\"
<<  Author << \"\\n\"
<< \"\\\\date{\"<< Date
<<\"}\\n\"
<< \"\\\\begin{document}\\n\"
<< \"\\\\maketitle\\n\";
```

```
214
215 file << \"\\\\section*{Variables Definition}\\n\";
216
217 for (int i = 0; i < NUM_OF_VARIABLES; i++) {
218 file << \"\\\\[\\n\" << var_def[i] << \" = \" << vdata[i] <<
        \"\\n\\\\]\\n\";
219 }
220
221 file << \"\\\\section*{Matrix Definition}\\n\";
222 for (int i = 0; i < NUM_OF_MATRIX_VARIABLES; i++) {
223 file << \"\\\\[\\n\" << matrix_def[i] << \" = \" << mdata[i
    ].printm() << \"\\n\\\\]\\n\";
224 }
225
226 file << \"\\\\section*{Formula Definition}\\n\";
227
228 file << \"\\\\begin{gather*}\\n\";
229
230 for (int i = 0; i < NUM_OF_FORMULAR_DEFINITION; i++) {
231 if (i != NUM_OF_FORMULAR_DEFINITION - 1) {
232 file << formular_def[i] << \"\\\\\\\\\\n\";
233 }
234 else {
235 file << formular_def[i] << \"\\n\";
236 }
237 }
238
239 file << \"\\\\end{gather*}\\n\";
240
241 file << \"\\\\section*{Logical Validation}\\n\";
242
243 for (int i = 0; i < NUM_OF_LOGICAL_VALIDATION; i++) {
244 file << \"\\\\[\\n\" << l_result[i] << \"\\n\\\\]\\n\";
245 }
246
247 file << \"\\\\section*{Formula Evaluation}\\n\";
248
249 for (int i = 0; i < NUM_OF_FORMULAR_EVALUATION ; i++) {
250 file << \"\\\\[\\n\" <<  c_result[i] << \"\\n\\\\]\\n\";
251 }
252
253 file << \"\\\\end{document}\\n\";
254
255 file.close();
256 return 0;
257 }
258 "
259 ;;
260
261
```

```
(* Utility Tools*)
(* *_to_string: Latex format *)
(* *_to_string_c: C++ format *)
(* *_to_string_p: Latex format for runtime-augument printing
    *)
(* *_to_string_interpreter: Interpreter format *)

let matrix_row_to_string_c (row: float array) : string =
    String.concat ", " (Array.to_list (Array.map (fun e ->
    float2str e) row))
;;
let matrix_to_string_c (matrix: matrix) : string = "matrix("
     ^ string_of_int (Array.length matrix) ^ ", " ^
    string_of_int (Array.length (matrix.(0))) ^ ", (const
    double []) {" ^
(String.concat ", " (Array.to_list (Array.map
    matrix_row_to_string_c matrix))) ^ "})"
;;

let matrix_row_to_string (row: float array) : string =
    String.concat " & " (Array.to_list (Array.map (fun e ->
    float2str e) row))
;;
let matrix_to_string (matrix: matrix) : string =
"\\begin{bmatrix}\n" ^
String.concat "\\\\\n" (Array.to_list (Array.map
    matrix_row_to_string matrix)) ^
"\n\\end{bmatrix}"
;;

(* expresion list to c++ string *)
let rec expr_list_to_string_c (para_list: expr list) :
    string = String.concat ", " (List.map (fun e -> fst(
    expr_to_string_c e)) para_list)

and
expr_to_string_c (expr : expr) : string * int = match expr
    with
Literal(f) -> (float2str f), 6
|   Id(s) -> s, 6
|   Assign(vid, e) -> (vid ^ "=" ^ (fst (expr_to_string_c e)
    )), 0
|   Binop(e1, op, e2) ->
let (s1, i1) = expr_to_string_c e1 in
let (s2, i2) = expr_to_string_c e2 in
(match op with
Add -> let (op_s, i) = "+", 3 in ((if i1 >= i then s1 else "
    (" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^ s2
     ^ ")")), i
```

```
295 | Sub -> let (op_s, i) = "-", 3 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
        s2 ^ ")")), i
296 | Mul -> let (op_s, i) = "*", 4 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
        s2 ^ ")")), i
297 | Div -> let (op_s, i) = "/", 4 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
        s2 ^ ")")), i
298 | Equal -> let (op_s, i) = "==", 1 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
299 | Neq -> let (op_s, i) = "!=", 1 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
300 | Less -> let (op_s, i) = "<", 2 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
301 | Leq -> let (op_s, i) = "<=", 2 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
302 | Greater -> let (op_s, i) = ">", 2 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
303 | Geq -> let (op_s, i) = ">=", 2 in ((if i1 >= i then s1
        else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
        " ^ s2 ^ ")")), i
304 | Pow -> let i = 5 in ( "pow(" ^ s1 ^ ", " ^ s2 ^ ")"), i
305 | DotMul -> let i = 4 in ( "DotMul(" ^ s1 ^ ", " ^ s2 ^ ")")
        , i
306 )
307 |   Sum(a, u, e) -> let vid, ex = match a with
308 Assign(vid, ex) -> vid, ex
309 | _ -> raise (Failure ("first parameter in sum should be
        assign"))
310 in
311 "[](int bottom, int top){double sum=0; for(int " ^ vid ^ "=
        bottom; " ^ vid ^ "<top+1;" ^ vid ^ "++) sum+=" ^ (fst(
        expr_to_string_c e))
312 ^ "; return sum;}(" ^ "(int)(" ^ (fst(expr_to_string_c ex))
        ^ "), " ^ "(int)(" ^ (fst(expr_to_string_c u)) ^ "))", 6
313 |   Prod(a, u, e) -> let vid, ex = match a with
314 Assign(vid, ex) -> vid, ex
315 | _ -> raise (Failure ("first parameter in prod should be
        assign"))
316 in
317 "[](int bottom, int top){double prod=1; for(int " ^ vid ^ "=
        bottom; " ^ vid ^ "<top+1;" ^ vid ^ "++) prod*=" ^ (fst(
        expr_to_string_c e))
```

```
318 ^ "; return prod;}(" ^ "(int)(" ^ (fst(expr_to_string_c ex))
        ^ "), " ^ "(int)(" ^ (fst(expr_to_string_c u)) ^ "))", 6
319 |   Call(f, para_list) -> f ^ "(" ^ (expr_list_to_string_c
        para_list) ^ ")", 6
320 |   MLiteral(m) -> (matrix_to_string_c m), 6
321 |   Uniop(op, e) ->
322 (
323 match op with
324 Trans -> "~" ^ "(" ^ (fst(expr_to_string_c e)) ^ ")", 6
325 | Sin   -> "sin " ^ "(" ^ (fst(expr_to_string_c e)) ^ ")", 6
326 | Cos   -> "cos " ^ "(" ^ (fst(expr_to_string_c e)) ^ ")", 6
327 | Log   -> "log10 " ^ "(" ^ (fst(expr_to_string_c e)) ^ ")",
        6
328 | Tan   -> "tan" ^ "(" ^ (fst(expr_to_string_c e)) ^ ")", 6
329 )
330 ;;
331
332 let expr_list_to_string_p (seperater: string) (para_list:
        expr list) : string = String.concat seperater (List.map (
        fun e -> "dtos(" ^ fst(expr_to_string_c e) ^ ")")
        para_list)
333 ;;
334
335 (* expresion list to latex string *)
336 let rec expr_list_to_string (para_list: expr list) : string
        = String.concat ", " (List.map (fun e -> fst(
        expr_to_string e)) para_list)
337
338 and
339 expr_to_string (expr : expr) : string * int = match expr
        with
340 Literal(f) -> (float2str f), 6
341 |   Id(s) -> s, 6
342 |   Assign(vid, e) -> (vid ^ "=" ^ (fst (expr_to_string e)))
        , 0
343 |   Binop(e1, op, e2) ->
344 let (s1, i1) = expr_to_string e1 in
345 let (s2, i2) = expr_to_string e2 in
346 (match op with
347 Add -> let (op_s, i) = "+", 3 in ((if i1 >= i then s1 else "
        (" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^ s2
        ^ ")")), i
348 | Sub -> let (op_s, i) = "-", 3 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
        s2 ^ ")")), i
349 | Mul -> let (op_s, i) = "*", 4 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
        s2 ^ ")")), i
350 | Div -> let (op_s, i) = "/", 4 in ((if i1 >= i then s1 else
        "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
```

```
    s2 ^ ")")), i
| Equal -> let (op_s, i) = "==", 1 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Neq -> let (op_s, i) = "!=", 1 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Less -> let (op_s, i) = "<", 2 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Leq -> let (op_s, i) = "<=", 2 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Greater -> let (op_s, i) = ">", 2 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Geq -> let (op_s, i) = ">=", 2 in ((if i1 >= i then s1
    else "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(
    " ^ s2 ^ ")")), i
| Pow -> let (op_s, i) = "^", 5 in ((if i1 >= i then s1 else
     "(" ^ s1 ^ ")") ^ op_s ^ "{" ^ s2 ^ "}"), i
| _ -> raise (Failure ("No matrix in logical expressions and
     function calls"))
)
|   Sum(a, u, e) -> "\\sum_{" ^ (fst(expr_to_string a)) ^ "
    }^{" ^ (fst(expr_to_string u)) ^ "}{" ^ (fst(
    expr_to_string e)) ^ "}", 1
|   Prod(a, u, e) -> "\\prod_{" ^ (fst(expr_to_string a)) ^
    "}^{" ^ (fst(expr_to_string u)) ^ "}{" ^ (fst(
    expr_to_string e)) ^ "}", 1
|   Uniop(op, e) ->
(
match op with
Sin   -> "\\sin" ^ "{" ^ (fst(expr_to_string e)) ^ "}", 1
| Cos   -> "\\cos" ^ "{" ^ (fst(expr_to_string e)) ^ "}", 1
| Log   -> "\\log_{10}" ^ "{" ^ (fst(expr_to_string e)) ^ "}
    ", 1
| Tan   -> "\\tan"      ^ "{" ^ (fst(expr_to_string e)) ^ "}
    ", 1
| _     -> raise (Failure ("No matrix in logical expressions
     and function calls"))
)
|   Call(f, para_list) -> f ^ "(" ^ (expr_list_to_string
    para_list) ^ ")", 6
|   _ -> raise (Failure ("No matrix in logical expressions
    and function calls"))
;;

let case_to_string (case: case) : string =
let expression = fst(expr_to_string case.expression) in
```

```
let condition = fst(expr_to_string case.condition) in
expression ^ " & " ^ condition
;;

let case_list_to_string (case_list: case list) : string =
    String.concat "\\\\\n" (List.map case_to_string case_list
    )
;;

let formular_to_string (formular : formular) : string =
let fname = formular.fname in
let parameter = expr_list_to_string formular.parameter in
let definition = match formular.definition with
Regular(e) -> fst(expr_to_string e)
|  Piecewise(case_list) -> "\\begin{cases}\n" ^ (
    case_list_to_string case_list) ^ "\n\\end{cases}"
in
fname ^ "(" ^ parameter ^ ")=" ^ definition
;;

let case_to_string_interpreter (case: case) : string =
let expression = fst(expr_to_string case.expression) in
let condition = fst(expr_to_string case.condition) in
"    " ^ expression ^ ", if " ^ condition ^ "."
;;

let case_list_to_string_interpreter (case_list: case list) :
     string = String.concat " Or\n" (List.map
    case_to_string_interpreter case_list)
;;

let formular_to_string_interpreter (formular : formular) :
    string =
let fname = formular.fname in
let parameter = expr_list_to_string formular.parameter in
let definition = match formular.definition with
Regular(e) -> fst(expr_to_string e)
|  Piecewise(case_list) -> "{\n" ^ (
    case_list_to_string_interpreter case_list) ^ "\n}"
in
fname ^ "(" ^ parameter ^ ") = " ^ definition
;;
```

Listing 8.5: interpret.ml

```
open Ast
open Header

module StringMap = Map.Make(String)
module StringSet = Set.Make(String)
```

```
6
7  (*
8  User-defined Map, key: (string * int)
9  For Formular Overloading
10 *)
11
12 module FunctionParamLen =
13 struct
14 type t = string * int
15 let compare (x1,x2) (y1,y2) =
16 if x1 < y1 then -1
17 else if x1 > y1 then 1
18 else if x2 < y2 then -1
19 else if x2 > y2 then 1
20 else 0
21 end ;;
22
23 module FPMap = Map.Make(FunctionParamLen)
24
25 (* Symbol table: Information about all the names in scope *)
26 type v_table = float StringMap.t
27
28 type f_table = formular FPMap.t
29
30 type m_table = matrix StringMap.t
31
32 type global_config = v_table * m_table
33
34 type rettype = Float of float | Matrix of matrix
35
36 (* Global mutable list,
37 information about formualr evalution instrcution
38 and   logical evalution instruction
39 *)
40 let fcall_strs = ref [];;
41 let lvali_strs = ref [];;
42
43 (* Get item in tripe-pairs *)
44 let first (x,_,_) = x;;
45 let second (_,y,_) = y;;
46 let third (_,_,z) = z;;
47
48 (* Matrix Multiplication *)
49 let matrix_constant_mul (lambda:float) (x:matrix)  : matrix
       =
50 let x_row = Array.length x in
51 if x_row = 0 then raise (Failure ("Invalid matrix with row#
       = 0")) else let x_col = Array.length x.(0) in
52 if x_col = 0 then raise (Failure ("Invalid matrix with col#
       = 0")) else
```

```
53 let z = Array.make_matrix x_row x_col 0. in
54 for i = 0 to x_row-1 do
55 for j = 0 to x_col-1 do
56 z.(i).(j) <- lambda *. x.(i).(j)
57 done
58 done;
59 (z)
60
61 (* Matrix Multiplication *)
62 let matrix_multiply (x:matrix) (y:matrix) : matrix =
63 let x_row = Array.length x and y_row = Array.length y in
64 if x_row = 0 then raise (Failure ("Invalid matrix with row#
       = 0")) else let x_col = Array.length x.(0) in
65 if y_row = 0 then raise (Failure ("Invalid matrix with row#
       = 0")) else let y_col = Array.length y.(0) in
66 if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid
       matrix with col# = 0")) else
67 if x_col <> y_row then raise (Failure ("Invalid matrix
       multiply ")) else
68 let z = Array.make_matrix x_row y_col 0. in
69 for i = 0 to x_row-1 do
70 for j = 0 to y_col-1 do
71 for k = 0 to y_row-1 do
72 z.(i).(j) <- z.(i).(j) +. x.(i).(k) *. y.(k).(j)
73 done
74 done
75 done;
76 (z)
77
78 (* Matrix Multiplication *)
79 let matrix_dot_multiply (x:matrix) (y:matrix) : matrix =
80 let x_row = Array.length x and y_row = Array.length y in
81 if x_row = 0 then raise (Failure ("Invalid matrix with row#
       = 0")) else let x_col = Array.length x.(0) in
82 if y_row = 0 then raise (Failure ("Invalid matrix with row#
       = 0")) else let y_col = Array.length y.(0) in
83 if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid
       matrix with col# = 0")) else
84 if (x_row <> y_row) || (x_col <> y_col) then raise (Failure
       ("Invalid matrix dot multiply ")) else
85 let z = Array.make_matrix x_row x_col 0. in
86 for i = 0 to x_row-1 do
87 for j = 0 to x_col-1 do
88 z.(i).(j) <- x.(i).(j) *. y.(i).(j)
89 done
90 done;
91 (z)
92
93 (* Matrix addition *)
94 let matrix_add (x:matrix) (y:matrix) : matrix =
```

```
95  let x_row = Array.length x and y_row = Array.length y in
96  if x_row = 0 then raise (Failure ("Invalid matrix with row#
        = 0")) else let x_col = Array.length x.(0) in
97  if y_row = 0 then raise (Failure ("Invalid matrix with row#
        = 0")) else let y_col = Array.length y.(0) in
98  if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid
        matrix with col# = 0")) else
99  if (x_row <> y_row) || (x_col <> y_col) then raise (Failure
        ("Invalid matrix dot multiply ")) else
100 let z = Array.make_matrix x_row x_col 0. in
101 for i = 0 to x_row-1 do
102 for j = 0 to x_col-1 do
103 z.(i).(j) <- x.(i).(j) +. y.(i).(j)
104 done
105 done;
106 (z)
107
108 (* Matrix subtraction *)
109 let matrix_sub (x:matrix) (y:matrix) : matrix =
110 let x_row = Array.length x and y_row = Array.length y in
111 if x_row = 0 then raise (Failure ("Invalid matrix with row#
        = 0")) else let x_col = Array.length x.(0) in
112 if y_row = 0 then raise (Failure ("Invalid matrix with row#
        = 0")) else let y_col = Array.length y.(0) in
113 if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid
        matrix with col# = 0")) else
114 if (x_row <> y_row) || (x_col <> y_col) then raise (Failure
        ("Invalid matrix dot multiply ")) else
115 let z = Array.make_matrix x_row x_col 0. in
116 for i = 0 to x_row-1 do
117 for j = 0 to x_col-1 do
118 z.(i).(j) <- x.(i).(j) -. y.(i).(j)
119 done
120 done;
121 (z)
122
123 (* Matrix Transpose *)
124 let matrix_transpose (x:matrix) : matrix =
125 let x_row = Array.length x in
126 if x_row = 0 then raise (Failure ("Invalid matrix with row#
        = 0")) else let x_col = Array.length x.(0) in
127 if x_col = 0 then raise (Failure ("Invalid matrix with col#
        = 0")) else
128 let z = Array.make_matrix x_col x_row 0. in
129 for i = 0 to x_col-1 do
130 for j = 0 to x_row-1 do
131 z.(i).(j) <- x.(j).(i)
132 done
133 done;
134 (z)
```

```
135
136 (*
137 Executing program, output result in stdout
138 @parameter : program : formulars * statements (defined in
        ast.ml)
139 @parameter : (title, author)
140 *)
141
142 let run ((formulars, statements) : program) (title, author)
        : unit =
143
144 (* Build a symbol table for function declarations *)
145 let func_decls : f_table = List.fold_left
146 (* Semantics checking: parameter names should be unique;
        function with same signature (name + # of parameters)
        should be unique *)
147 (fun functions func -> ignore(List.fold_left (fun param_map
        expr -> match expr with
148 Id(n) -> if StringSet.mem n param_map then raise (Failure ("
        Parameter redefinition of " ^ n)) else
149 (StringSet.add n param_map)
150 | _ -> raise (Failure ("Parameter is not an ID"))
151 )
152 StringSet.empty func.parameter);
153 if FPMap.mem (func.fname, List.length func.parameter)
        functions then raise (Failure ("Function " ^ func.fname ^
         " with " ^ string_of_int (List.length func.parameter) ^
        " parameters already exists."))
154 else FPMap.add (func.fname, List.length func.parameter) func
         functions)
155 FPMap.empty formulars
156 in
157
158 let func_name = FPMap.fold (fun k v set -> StringSet.add (
        fst(k)) set) func_decls StringSet.empty
159 in
160
161 (* Initial Empty Matrix table, id(string) -> val(matrix) *)
162 let global_matrices : m_table = StringMap.empty
163 in
164
165 (* Initial Empty Variable table, id(string) -> val(Float) *)
166 let global_vars : v_table = StringMap.empty
167 in
168
169 (* Formular execution utilities *)
170 let rec exec_formular (fdecl : formular)(actuals : float
        list)(global_vars : v_table) : rettype * v_table =
171 let get_name (name:expr) = match name with
172 Id(n) -> n
```

```
173 |   _ -> raise (Failure ("Parameter is not an ID"))
174 in
175
176 let find_case (config : rettype * v_table * bool) (case :
        case) : bool =
177 let global_vars = second(config) in
178 let v, (_, _) = eval (global_vars, global_matrices) case.
        condition in
179 match v with
180 Float(f) -> (f > 0.)
181 | _ -> raise (Failure ("Matrix is not allowed in case
        condition"))
182
183 in
184
185 let local_vars : v_table = List.fold_left2 (fun local_vars
        actual name -> StringMap.add (get_name name) actual
        local_vars) global_vars actuals fdecl.parameter
186 in
187 match fdecl.definition with
188 Regular(e) -> let (result, (local_vars, global_matrices)) =
        eval (local_vars, global_matrices) e in (result,
        global_vars) (* Note: global_matrix is empty *)
189 |   Piecewise(case_list) -> let (result, local_vars,
        is_found) = List.fold_left
190 (fun (config : rettype * v_table * bool) (case : case) -> if
         (not (third(config)) && find_case config case) then (fst
        (eval (local_vars, global_matrices) case.expression),
        second(config), true) else (config) )
191 (Float(0.), local_vars, false) case_list
192 in
193 if is_found then (result, global_vars) else raise (Failure (
        "Piecewise not complete"))
194
195 and
196
197 (* Main evaluation function *)
198 eval ((global_vars : v_table), (global_matrices : m_table))
        (exp : expr) : rettype * global_config = match exp with
199 (* Return Constant  *)
200 Literal(f) -> Float(f), (global_vars, global_matrices)
201 |   MLiteral(m) -> Matrix(m), (global_vars, global_matrices)
202 (* first try to find in float vars, then try to find in
        matrix vars *)
203 |   Id(s) -> if StringMap.mem s global_vars then
204 Float((StringMap.find s global_vars)), (global_vars,
        global_matrices)
205 else if StringMap.mem s global_matrices then
206 Matrix((StringMap.find s global_matrices)), (global_vars,
        global_matrices)
```

```
207 else raise (Failure ("Undeclared identifier " ^ s))
208 |   Assign(vid, e) ->
209 if StringSet.mem vid func_name then raise (Failure ("Cannot
        assign a new var with the same name of a formular"))
210 else
211 let (v, (global_vars, global_matrices)) = eval (global_vars,
        global_matrices) e in
212 (match v with
213 Float(f) -> if StringMap.mem vid global_matrices then raise
        (Failure ("Cannot assign a new var with the same name of
        a matrix")) else
214 (Float(f), ((StringMap.add vid f global_vars),
        global_matrices))
215 | Matrix(m) -> if StringMap.mem vid global_vars then raise (
        Failure ("Cannot assign a var with a matrix")) else
216 Matrix(m), (global_vars, (StringMap.add vid m
        global_matrices))
217 )
218 |   Sum(a, u, e) -> let vid, ex = match a with
219 Assign(vid, ex) -> vid, ex
220 | _ -> raise (Failure ("first parameter in sum should be
        assign"))
221 in
222 let v1, (global_vars, global_matrices) = eval (global_vars,
        global_matrices) ex in
223 let v2, (global_vars, global_matrices) = eval (global_vars,
        global_matrices) u in
224 (match (v1,v2) with
225 (Float(f1), Float(f2)) ->   let sum = ref 0. in
226 for i = (int_of_float f1) to (int_of_float f2) do
227 let global_vars = StringMap.add vid (float_of_int i)
        global_vars in
228 let v3, (global_vars, global_matrices) = eval (global_vars,
        global_matrices) e in
229 let ret = match v3 with
230 Float(f3) -> f3
231 | _ -> raise (Failure ("no matrix in sum"))
232 in
233 sum := !sum +. ret
234 done;
235 Float(!sum), (global_vars, global_matrices)
236 | _ -> raise (Failure ("no matrix in sum"))
237 )
238 |   Prod(a, u, e) -> let vid, ex = match a with
239 Assign(vid, ex) -> vid, ex
240 | _ -> raise (Failure ("first parameter in prod should be
        assign"))
241 in
242 let v1, (global_vars, global_matrices) = eval (global_vars,
        global_matrices) ex in
```

```
let v2, (global_vars, global_matrices) = eval (global_vars, global_matrices) u in
(match (v1,v2) with
(Float(f1), Float(f2)) ->   let prod = ref 1. in
for i = (int_of_float f1) to (int_of_float f2) do
let global_vars = StringMap.add vid (float_of_int i) global_vars in
let v3, (global_vars, global_matrices) = eval (global_vars, global_matrices) e in
let ret = match v3 with
Float(f3) -> f3
| _ -> raise (Failure ("no matrix in prod"))
in
prod := !prod *. ret
done;
Float(!prod), (global_vars, global_matrices)
| _ -> raise (Failure ("no matrix in prod"))
)
|   Uniop(op, e) -> let (r, (global_vars, global_matrices)) = eval (global_vars, global_matrices) e in
(
match r with
Matrix(m) ->( match op with
Trans -> (Matrix(matrix_transpose m), (global_vars, global_matrices))
| _    -> raise (Failure ("Cannot perform other unit operator on matrix"))
)
| Float(f) -> (
match op with
Sin -> (Float(sin f), (global_vars, global_matrices))
| Cos -> ((Float(cos f), (global_vars, global_matrices)))
| Log -> ((Float(log10 f), (global_vars, global_matrices)))
| Tan -> ((Float(tan f), (global_vars, global_matrices)))
| _ -> raise (Failure ("Unsupported unit operator on float number")) (* Future work: add more unit operator function *)
)
)
|   Binop(e1, op, e2) ->
let v1, (global_vars, global_matrices) = eval (global_vars, global_matrices) e1 in
let v2, (global_vars, global_matrices) = eval (global_vars, global_matrices) e2 in
let boolean i = if i then 1. else 0. in
(match (v1,v2) with
(Float(f1),  Float(f2)) ->      (match op with
Add -> Float(f1 +. f2)
| Sub -> Float(f1 -. f2)
| Mul -> Float(f1 *. f2)
```

```
283 | Div -> if f2 = 0. then raise (Failure ("Division by 0!"))
        else
284 Float(f1 /. f2)
285 | Equal -> Float(boolean (f1 = f2))
286 | Neq -> Float(boolean (f1 <> f2))
287 | Less -> Float(boolean (f1 < f2))
288 | Leq -> Float(boolean (f1 <= f2))
289 | Greater -> Float(boolean (f1 > f2))
290 | Geq -> Float(boolean (f1 >= f2))
291 | Pow -> Float(f1 ** f2)
292 | _ -> raise (Failure ("Matirx op can't apply on float"))
293 ), (global_vars, global_matrices)
294 | (Float(f1), Matrix(m1)) -> (match op with
295 Mul -> (Matrix(matrix_constant_mul f1 m1 ), (global_vars,
        global_matrices))
296 | _ -> raise (Failure ("Float can only apply Mul with matrix
        "))
297 )
298 | (Matrix(m1), Float(f1)) ->  raise (Failure ("Matirx op
        Float not allowed"))
299 | (Matrix(m1), Matrix(m2)) -> (match op with
300 Mul -> (Matrix(matrix_multiply m1 m2), (global_vars,
        global_matrices)) (* Matrix Multiplication operation*)
301 | DotMul -> (Matrix(matrix_dot_multiply m1 m2), (global_vars
        , global_matrices)) (* Matrix DotMul operation*)
302 | Add -> (Matrix(matrix_add m1 m2), (global_vars,
        global_matrices)) (* Matrix add operation*)
303 | Sub -> (Matrix(matrix_sub m1 m2), (global_vars,
        global_matrices)) (* Matrix sub operation*)
304 | _ -> raise (Failure ("Matirx op Matrix not allowed"))
305 )
306 )
307
308 |   Call(f, para_list) ->
309 let fdecl =
310 try
311 FPMap.find (f, List.length para_list) func_decls
312 with Not_found -> raise (Failure ("Undefined function " ^ f)
        )
313 in
314 let (actuals: float list), (global_vars, global_matrices) =
        List.fold_left
315 (
316 fun (actuals, (global_vars, global_matrices)) actual ->
317 let v, (global_vars, global_matrices) = eval (global_vars,
        global_matrices) actual in
318 match v with
319 Float(f) -> f :: actuals, (global_vars, global_matrices)
320 | Matrix(m) -> raise (Failure ("Formular can not take in
        matrix"))
```

```
)
([], (global_vars, global_matrices)) (List.rev para_list)
in let (r, v) = exec_formular fdecl actuals global_vars
in let f = match r with Matrix(m) -> raise (Failure ("Matirx
    not allowed in function return type")) | Float(f) -> f
in (Float(f), (v, global_matrices))

in

(* Capture function calls and logical validations in top-
    level and handle printing *)
let eval_top ((global_vars : v_table), (global_matrices :
    m_table)) (exp : expr) : rettype * global_config =
match exp with
Call(f, para_list) ->
let (actuals: float list), (_, _) = List.fold_left
(
fun (actuals, (global_vars, global_matrices)) actual ->
let v, (global_vars, global_matrices) = eval (global_vars,
    global_matrices) actual in
match v with
Float(f) -> f :: actuals, (global_vars, global_matrices)
| Matrix(m) -> raise (Failure ("Formular can not take in
    matrix"))
)
([], (global_vars, global_matrices)) (List.rev para_list)
in
let (r, global_config) =  (eval (global_vars,
    global_matrices) exp) in
let fcall_str = (f ^ "(" ^ (String.concat "," (List.map
    float2str actuals)) ^ ") = " ^  float2str (match r with
    Float(r) -> r | _ -> raise (Failure ("You shouldn't see
    this error")))) in
(fcall_strs := fcall_str :: !fcall_strs);
(r,global_config)
| Binop(e1, op, e2) ->
(match op with
Equal | Neq | Less | Leq | Greater | Geq  ->
let (r, global_config) = (eval (global_vars, global_matrices
    ) exp) in
let lvali_str = (fst(expr_to_string exp) ^ " is " ^ (match r
     with Float(f) -> if f > 0. then "true" else "false" | _
    -> raise (Failure ("You shouldn't see this error")))) in
(lvali_strs := lvali_str :: !lvali_strs);
(r, global_config)
| _ -> eval (global_vars, global_matrices) exp
)
| _ -> eval (global_vars, global_matrices) exp
in

```

```
359 |
360 | let process_stmt (global_config:global_config) (statement :
    |     statement) =
361 | match statement with
362 | Seq(expr_list) -> (List.fold_left
363 | (fun a b -> snd(eval_top a b)) global_config expr_list)
364 | in
365 |
366 | (* Final Report Printing *)
367 | let _=  print_endline (title ^ " " ^ author) in
368 | let _ = print_endline "
    |     ----------------------------------------\n
    |     Formular Definitions\n
    |     ----------------------------------------" in
369 | let _ = List.iter (fun f -> print_endline(
    |     formular_to_string_interpreter f)) formulars in
370 | let (global_vars, global_matrices) = List.fold_left
    |     process_stmt (global_vars, global_matrices) statements in
371 | let _ = print_endline "
    |     ----------------------------------------\n
    |     Formular Evaluation\n
    |     ----------------------------------------" in
372 | let _ = List.iter print_endline (List.rev !fcall_strs) in
373 | let _ = print_endline "
    |     ----------------------------------------\n
    |     Logical Validation\n
    |     ----------------------------------------" in
374 | let _ = List.iter print_endline (List.rev !lvali_strs) in
375 | let _ = print_endline "
    |     ----------------------------------------\n
    |     Variable Definitions\n
    |     ----------------------------------------" in
376 | StringMap.iter (fun key value -> print_endline(key ^ " = " ^
    |      float2str(value))) global_vars;
377 | print_endline "----------------------------------------\n
    |               Matrix Definitions\n
    |     ----------------------------------------";
378 | StringMap.iter (fun key value -> print_endline(key ^ " = " ^
    |      Ast.matrix_decl_s(value))) global_matrices
```

Listing 8.6: compile.ml

```
1 | open Header
2 | open Ast
3 | open Printf (* Header for write file *)
4 |
5 | module StringSet = Set.Make(String)
6 |
7 | module StringMap = Map.Make(String)
8 |
```

```
9  (*
10 User-defined Map, key: (string * int)
11 For Formular Overloading
12 *)
13
14 module FunctionParamLen =
15 struct
16 type t = string * int
17 let compare (x1,x2) (y1,y2) =
18 if x1 < y1 then -1
19 else if x1 > y1 then 1
20 else if x2 < y2 then -1
21 else if x2 > y2 then 1
22 else 0
23 end ;;
24
25 module FPMap = Map.Make(FunctionParamLen)
26
27 (* Symbol table: Information about all the names in scope *)
28
29 type f_table = formular FPMap.t
30
31 type v_table = string list StringMap.t
32
33 type rettype = Float| Matrix
34
35 (*
36 Translate program into C++ code
37 @parameter : program : formulars * statements (defined in
       ast.ml)
38 @parameter : (title, author, cpp file name, latex file name)
39 *)
40
41 let translate ((formulars, statements) : program) (title,
       author, cppfile, lfile): unit =
42
43 (* global float variable table *)
44 let v_set = StringSet.empty in
45
46 (* global matrix variable table *)
47 let m_set = StringSet.empty in
48
49 (* # of logical expressions *)
50 let l_count = 0 in
51
52 (* # of function calls *)
53 let c_count = 0 in
54
55 (* List of instructions in C++ *)
56 let i_list = [] in
```

```
57
58 (* List of function definitions *)
59 let f_list = [] in
60
61 (* List of function declarations *)
62 let f_decl_list = [] in
63
64 (* Global variables used in function *)
65 let func_vars : v_table = StringMap.empty in
66
67 (* Function map *)
68 let func_decls : f_table = List.fold_left
69 (fun functions func -> ignore(List.fold_left (fun param_map
       expr -> match expr with
70         Id(n) -> if StringMap.mem n param_map then raise (
              Failure ("Parameter redefinition of " ^ n)) else
71          (StringMap.add n 0 param_map)
72         | _ -> raise (Failure ("Parameter is not an ID"))
73 )
74 StringMap.empty func.parameter);
75 if FPMap.mem (func.fname, List.length func.parameter)
       functions then raise (Failure ("Function " ^ func.fname ^
        " with " ^ string_of_int (List.length func.parameter) ^
       " parameters already exists."))
76 else FPMap.add (func.fname, List.length func.parameter) func
        functions)
77 FPMap.empty formulars
78 in
79
80 let func_name = FPMap.fold (fun k v set -> StringSet.add (
       fst(k)) set) func_decls StringSet.empty
81 in
82
83 (* Evaluation inside a function *)
84 let rec formular_eval (v_set, g_set) (expr : expr) =
85 (match expr with
86 Call(f, para_list) -> let fdecl =
87 try
88 FPMap.find (f, List.length para_list) func_decls
89 with Not_found -> raise (Failure ("Undefined function " ^ f)
       )
90 in
91 if (List.length fdecl.parameter)<>(List.length para_list)
       then raise (Failure ("Unmatched parameters"))
92 else
93 let (v_set, g_set) = List.fold_left
94 (fun (v_set, g_set) actual ->
95 let v, (v_set, g_set) = formular_eval (v_set, g_set) actual
       in
96 match v with
```

```
97  Float -> (v_set, g_set)
98  | Matrix -> raise (Failure ("Formular can not take in matrix
        "))
99  )
100 (v_set, g_set) (List.rev para_list)
101 in
102 Float, (v_set, g_set)
103 | Binop(e1, op, e2) ->  let v1, (v_set, g_set) =
        formular_eval (v_set, g_set) e1 in
104 let v2, (v_set, g_set) = formular_eval (v_set, g_set) e2 in
105 (match (v1,v2) with
106 (Float, Float) ->
107 (match op with
108 Equal -> Float, (v_set, g_set)
109 | Neq -> Float, (v_set, g_set)
110 | Less -> Float, (v_set, g_set)
111 | Leq -> Float, (v_set, g_set)
112 | Greater -> Float, (v_set, g_set)
113 | Geq -> Float, (v_set, g_set)
114 | Add -> Float, (v_set, g_set)
115 | Sub -> Float, (v_set, g_set)
116 | Mul -> Float, (v_set, g_set)
117 | Div -> Float, (v_set, g_set)
118 | Pow -> Float, (v_set, g_set)
119 | _ -> raise (Failure ("Matirx op can't apply on Float"))
120 )
121 |   _ -> raise (Failure ("No matrix in formular"))
122 )
123 | Sum(a, u, e) -> let vid, ex = match a with
124 Assign(vid, ex) -> vid, ex
125 | _ -> raise (Failure ("first parameter in sum should be
        assign"))
126 in
127 let v1, (v_set, g_set) = formular_eval (v_set, g_set) ex in
128 let v2, (v_set, g_set) = formular_eval (v_set, g_set) u in
129 let tmp_v_set = if StringSet.mem vid v_set then v_set else
        StringSet.add vid v_set in
130 let v3, (tmp_v_set, g_set) = formular_eval (tmp_v_set, g_set
        ) e in
131 (match (v1,v2,v3) with
132 (Float, Float, Float) -> let v_set = if StringSet.mem vid
        v_set then tmp_v_set else StringSet.remove vid tmp_v_set
        in
133 Float, (v_set, g_set)
134 | _ -> raise (Failure ("no matrix in sum"))
135 )
136 | Prod(a, u, e) -> let vid, ex = match a with
137 Assign(vid, ex) -> vid, ex
138 | _ -> raise (Failure ("first parameter in prod should be
        assign"))
```

```
in
let v1, (v_set, g_set) = formular_eval (v_set, g_set) ex in
let v2, (v_set, g_set) = formular_eval (v_set, g_set) u in
let tmp_v_set = if StringSet.mem vid v_set then v_set else
    StringSet.add vid v_set in
let v3, (tmp_v_set, g_set) = formular_eval (tmp_v_set, g_set
    ) e in
(match (v1,v2,v3) with
(Float, Float, Float) -> let v_set = if StringSet.mem vid
    v_set then tmp_v_set else StringSet.remove vid tmp_v_set
    in
Float, (v_set, g_set)
| _ -> raise (Failure ("no matrix in prod"))
)
| Uniop(op, e) -> let v, (v_set, g_set) = formular_eval (
    v_set, g_set) e in
( match v with
Float -> (
match op with
Sin -> Float, (v_set, g_set)
| Cos -> Float, (v_set, g_set)
| Log -> Float, (v_set, g_set)
| Tan -> Float, (v_set, g_set)
| _   -> raise (Failure ("Un-supported float unit operator")
    )
)
| _   ->  raise (Failure ("No matrix in formular"))
)
| Literal(f) -> Float, (v_set, g_set)
| Id(s) -> if StringSet.mem s v_set then
Float, (v_set, g_set)
else let g_set = StringSet.add s g_set in
Float, (v_set, g_set)
| _ -> raise (Failure ("Invalid expression in formular"))
)
in

let process_formular (func_vars, f_list, f_decl_list) (fdecl
     : formular) =

let get_name (name:expr) = match name with
Id(n) -> n
|   _ -> raise (Failure ("Parameter is not an ID"))
in

let local_set = List.fold_left
(fun set expr -> StringSet.add (get_name expr) set)
StringSet.empty fdecl.parameter
in

```

```
182 let process_case (v_set, g_set) (case : case) = let v, (
        v_set, g_set) = formular_eval (v_set, g_set) case.
        expression in
183 let v, (v_set, g_set) = formular_eval (v_set, g_set) case.
        condition in
184 let ret = "if(" ^ fst(expr_to_string_c case.condition) ^ ")
        return "
185 ^ fst(expr_to_string_c case.expression) ^ ";" in
186 (v_set, g_set), ret
187 in
188
189
190 match fdecl.definition with
191 Regular(e) -> let v, (v_set, g_set) = formular_eval (
        local_set, StringSet.empty) e in
192 let func_vars = StringMap.add fdecl.fname (StringSet.
        elements g_set) func_vars in
193 let decl = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        " (List.map (fun e -> "double " ^ fst(expr_to_string_c e)
        ) fdecl.parameter) ^ ");" in
194 let ret = "double " ^ fdecl.fname ^ "(" ^ String.concat ", "
         (List.map (fun e -> "double " ^ fst(expr_to_string_c e))
         fdecl.parameter) ^ "){\n\t"
195 ^ "return " ^ fst(expr_to_string_c e) ^ ";\n}" in
196 func_vars, ret::f_list, decl::f_decl_list
197
198 | Piecewise(case_list) -> let (v_set, g_set), tmp = List.
        fold_left (fun ((v_set, g_set), ret) case ->
199 let (v_set, g_set), r = process_case (v_set, g_set) case in
200 (v_set, g_set), ret ^ "\n\t" ^ r)
201 ((local_set, StringSet.empty), "") case_list
202 in
203 let func_vars = StringMap.add fdecl.fname (StringSet.
        elements g_set) func_vars in
204 let decl = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        " (List.map (fun e -> "double " ^ fst(expr_to_string_c e)
        ) fdecl.parameter) ^ ");" in
205 let ret = "double " ^ fdecl.fname ^ "(" ^  String.concat ",
        " (List.map (fun e -> "double " ^ fst(expr_to_string_c e)
        ) fdecl.parameter) ^ "){"
206 ^ tmp ^ "\n\tthrow std::runtime_error(\"Illegal parameter in
         piecewise function " ^ fdecl.fname ^ "\");\n}" in
207 func_vars, ret::f_list, decl::f_decl_list
208
209 in
210
211 let (func_vars, f_list, f_decl_list) =
212 List.fold_left process_formular (func_vars, f_list,
        f_decl_list) formulars
213 in
```

```
214 (* Normal evaluation *)
215 let rec eval (v_set, m_set, l_count, c_count, i_list) (expr
        : expr) =
216 (match expr with
217 Call(f, para_list) -> let fdecl =
218 try
219 FPMap.find (f, List.length para_list) func_decls
220 with Not_found -> raise (Failure ("Undefined function " ^ f)
        )
221 in
222 let g_list = StringMap.find f func_vars
223 in
224 List.iter (fun g -> if StringSet.mem g v_set then () else
        raise (Failure ("Uninitialized global variable " ^ g)) )
        g_list;
225 if (List.length fdecl.parameter)<>(List.length para_list)
        then raise (Failure ("Unmatched parameters"))
226 else
227 let (v_set, m_set, l_count, c_count, i_list) = List.
        fold_left
228 (fun (v_set, m_set, l_count, c_count, i_list) actual ->
229 let v, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) actual in
230 match v with
231 Float -> (v_set, m_set, l_count, c_count, i_list)
232 | Matrix -> raise (Failure ("Formular can not take in matrix
        "))
233 )
234 (v_set, m_set, l_count, c_count, i_list) (List.rev para_list
        )
235 in
236 Float, (v_set, m_set, l_count, c_count, i_list)
237 | Binop(e1, op, e2) ->  let v1, (v_set, m_set, l_count,
        c_count, i_list) = eval (v_set, m_set, l_count, c_count,
        i_list) e1 in
238 let v2, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) e2 in
239 (match (v1,v2) with
240 (Float, Float) ->
241 (match op with
242 Equal -> Float, (v_set, m_set, l_count, c_count, i_list)
243 | Neq -> Float, (v_set, m_set, l_count, c_count, i_list)
244 | Less -> Float, (v_set, m_set, l_count, c_count, i_list)
245 | Leq -> Float, (v_set, m_set, l_count, c_count, i_list)
246 | Greater -> Float, (v_set, m_set, l_count, c_count, i_list)
247 | Geq -> Float, (v_set, m_set, l_count, c_count, i_list)
248 | Add -> Float, (v_set, m_set, l_count, c_count, i_list)
249 | Sub -> Float, (v_set, m_set, l_count, c_count, i_list)
250 | Mul -> Float, (v_set, m_set, l_count, c_count, i_list)
251 | Div -> Float, (v_set, m_set, l_count, c_count, i_list)
```

```
252 | Pow -> Float, (v_set, m_set, l_count, c_count, i_list)
253 | _ -> raise (Failure ("Matirx op can't apply on Float"))
254 )
255 |   (Float, Matrix) ->
256 (match op with
257 Mul -> Matrix, (v_set, m_set, l_count, c_count, i_list)
258 | _ -> raise (Failure ("Float can only apply Mul with Matrix
        "))
259 )
260 |   (Matrix, Float) -> raise (Failure ("Matirx op Float not
        allowed"))
261 |   (Matrix, Matrix) ->
262 (match op with
263 Mul -> Matrix, (v_set, m_set, l_count, c_count, i_list)
264 | DotMul -> Matrix, (v_set, m_set, l_count, c_count, i_list)
265 | Add -> Matrix, (v_set, m_set, l_count, c_count, i_list)
266 | Sub -> Matrix, (v_set, m_set, l_count, c_count, i_list)
267 | _ -> raise (Failure ("Undefined Matrix operation"))
268 )
269 )
270 | Sum(a, u, e) -> let vid, ex = match a with
271 Assign(vid, ex) -> vid, ex
272 | _ -> raise (Failure ("first parameter in sum should be
        assign"))
273 in
274 let v1, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) ex in
275 let v2, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) u in
276 let tmp_v_set = if StringSet.mem vid v_set then v_set else
        StringSet.add vid v_set in
277 let v3, (tmp_v_set, m_set, l_count, c_count, i_list) = eval
        (tmp_v_set, m_set, l_count, c_count, i_list) e in
278 (match (v1,v2,v3) with
279 (Float, Float, Float) -> let v_set = if StringSet.mem vid
        v_set then tmp_v_set else StringSet.remove vid tmp_v_set
        in
280 Float, (v_set, m_set, l_count, c_count, i_list)
281 | _ -> raise (Failure ("no matrix in sum"))
282 )
283 | Prod(a, u, e) -> let vid, ex = match a with
284 Assign(vid, ex) -> vid, ex
285 | _ -> raise (Failure ("first parameter in prod should be
        assign"))
286 in
287 let v1, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) ex in
288 let v2, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) u in
```

```
289 let tmp_v_set = if StringSet.mem vid v_set then v_set else
        StringSet.add vid v_set in
290 let v3, (tmp_v_set, m_set, l_count, c_count, i_list) = eval
        (tmp_v_set, m_set, l_count, c_count, i_list) e in
291 (match (v1,v2,v3) with
292 (Float, Float, Float) -> let v_set = if StringSet.mem vid
        v_set then tmp_v_set else StringSet.remove vid tmp_v_set
        in
293 Float, (v_set, m_set, l_count, c_count, i_list)
294 | _ -> raise (Failure ("no matrix in prod"))
295 )
296 | Literal(f) -> Float, (v_set, m_set, l_count, c_count,
        i_list)
297 | MLiteral(m) -> Matrix, (v_set, m_set, l_count, c_count,
        i_list)
298 | Id(s) -> if StringSet.mem s v_set then
299 Float, (v_set, m_set, l_count, c_count, i_list)
300 else if StringSet.mem s m_set then
301 Matrix, (v_set, m_set, l_count, c_count, i_list)
302 else raise (Failure ("Undeclared identifier " ^ s))
303 | Uniop(op, e) -> let v, (v_set, m_set, l_count, c_count,
        i_list) = eval (v_set, m_set, l_count, c_count, i_list) e
         in
304 (match v with
305 Matrix ->
306 ( match op with
307 Trans -> Matrix, (v_set, m_set, l_count, c_count, i_list)
308 | _ -> raise (Failure ("Un-supported matrix unit operator"))
309 )
310 | Float ->
311 ( match op with
312 Sin -> Float, (v_set, m_set, l_count, c_count, i_list)
313 | Cos -> Float, (v_set, m_set, l_count, c_count, i_list)
314 | Log -> Float, (v_set, m_set, l_count, c_count, i_list)
315 | Tan -> Float, (v_set, m_set, l_count, c_count, i_list)
316 | _   -> raise (Failure ("Un-supported float unit operator")
        )
317 )
318 )
319 | Assign(vid, e) ->
320 if StringSet.mem vid func_name then raise (Failure ("Cannot
        assign a new var with the same name of a formular"))
321 else
322 let v, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) e in
323 (match v with
324 Float -> if StringSet.mem vid m_set then raise (Failure ("
        Cannot assign a new var with the same name of a matrix"))
         else
325 let v_set = StringSet.add vid v_set in
```

```
326 Float, (v_set, m_set, l_count, c_count, i_list)
327
328 | Matrix -> if StringSet.mem vid v_set then raise (Failure (
        "Cannot assign a var with a matrix")) else
329 let m_set = StringSet.add vid m_set in
330 Matrix, (v_set, m_set, l_count, c_count, i_list)
331 )
332 )
333 in
334
335 (* Capture function calls and logical validations in top-
        level and handle printing *)
336 let eval_top (v_set, m_set, l_count, c_count, i_list) (expr
        : expr) =
337 (match expr with
338 Call(f, para_list) -> let fdecl =
339 try
340 FPMap.find (f, List.length para_list) func_decls
341 with Not_found -> raise (Failure ("Undefined function " ^ f)
        )
342 in
343 let g_list = StringMap.find f func_vars
344 in
345 List.iter (fun g -> if StringSet.mem g v_set then () else
        raise (Failure ("Uninitialized global variable " ^ g)) )
        g_list;
346 if (List.length fdecl.parameter)<>(List.length para_list)
        then raise (Failure ("Unmatched parameters"))
347 else
348 let (v_set, m_set, l_count, c_count, i_list) = List.
        fold_left
349 (fun (v_set, m_set, l_count, c_count, i_list) actual ->
350 let v, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) actual in
351 match v with
352 Float -> (v_set, m_set, l_count, c_count, i_list)
353 | Matrix -> raise (Failure ("Formular can not take in matrix
        "))
354 )
355 (v_set, m_set, l_count, c_count, i_list) (List.rev para_list
        )
356 in
357
358 let display = if (List.length para_list)>0 then
359 "string(\"\")" ^ "+\"" ^  f ^ "(" ^ String.escaped(
        expr_list_to_string para_list) ^ ")=" ^ f ^ "(\"+"
360 ^ (expr_list_to_string_p "+\", \"+" para_list) ^ "+\")=\"+
        dtos(" ^ f ^ "("
361 ^ (expr_list_to_string_c para_list) ^ "))"
```

```
362 else "string(\"\")" ^ "+\"" ^  f ^ "(" ^ String.escaped(
        expr_list_to_string para_list) ^ ")=" ^ f ^ "(\""
363 ^ (expr_list_to_string_p "+\", \"+" para_list) ^ "+\")=\"+
        dtos(" ^ f ^ "("
364 ^ (expr_list_to_string_c para_list) ^ "))"
365 in
366 let c_count = c_count + 1
367 in
368 let instruction = "c_result[" ^ (string_of_int (c_count-1))
        ^ "]=" ^ display ^ ";"
369 in
370 let i_list = instruction :: i_list
371 in
372 (v_set, m_set, l_count, c_count, i_list)
373
374 | Binop(e1, op, e2) -> let v1, (v_set, m_set, l_count,
        c_count, i_list) = eval (v_set, m_set, l_count, c_count,
        i_list) e1 in
375 let v2, (v_set, m_set, l_count, c_count, i_list) = eval (
        v_set, m_set, l_count, c_count, i_list) e2 in
376 (match (v1,v2) with
377 (Float, Float) -> let display_first = "string(\"\")" ^ "+\""
         ^ String.escaped(fst(expr_to_string expr)) ^ " \"" in
378 let display_left = "dtos(" ^ fst(expr_to_string_c e1) ^ ")"
        in
379 let display_right = "dtos(" ^ fst(expr_to_string_c e2) ^ ")"
         in
380 let display_last = "(" ^ fst(expr_to_string_c expr) ^ "?" ^
        "\" true\":\" false\"" ^ ")" in
381 (match op with
382 Equal -> let display = display_first ^ "+" ^ "\"\\\\
        Rightarrow \"" ^ "+" ^ display_left ^ "+\"==\"+" ^
        display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
        display_last in
383 let l_count = l_count + 1 in
384 let instruction = "l_result[" ^ (string_of_int (l_count-1))
        ^ "]=" ^ display ^ ";" in
385 let i_list = instruction :: i_list in
386 (v_set, m_set, l_count, c_count, i_list)
387 | Neq -> let display = display_first ^ "+" ^ "\"\\\\
        Rightarrow \"" ^ "+" ^ display_left ^ "+\"!=\"+" ^
        display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
        display_last in
388 let l_count = l_count + 1 in
389 let instruction = "l_result[" ^ (string_of_int (l_count-1))
        ^ "]=" ^ display ^ ";" in
390 let i_list = instruction :: i_list in
391 (v_set, m_set, l_count, c_count, i_list)
392 | Less -> let display = display_first ^ "+" ^ "\"\\\\
        Rightarrow \"" ^ "+" ^ display_left ^ "+\"<\"+" ^
```

```
    display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
    display_last in
let l_count = l_count + 1 in
let instruction = "l_result[" ^ (string_of_int (l_count-1))
    ^ "]=" ^ display ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Leq -> let display = display_first ^ "+" ^ "\"\\\\
    Rightarrow \"" ^ "+" ^ display_left ^ "+\"<=\"+" ^
    display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
    display_last in
let l_count = l_count + 1 in
let instruction = "l_result[" ^ (string_of_int (l_count-1))
    ^ "]=" ^ display ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Greater -> let display = display_first ^ "+" ^ "\"\\\\
    Rightarrow \"" ^ "+" ^ display_left ^ "+\">\"+" ^
    display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
    display_last in
let l_count = l_count + 1 in
let instruction = "l_result[" ^ (string_of_int (l_count-1))
    ^ "]=" ^ display ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Geq -> let display = display_first ^ "+" ^ "\"\\\\
    Rightarrow \"" ^ "+" ^ display_left ^ "+\">=\"+" ^
    display_right ^ "+" ^ "\" \\\\Rightarrow\"" ^ "+" ^
    display_last in
let l_count = l_count + 1 in
let instruction = "l_result[" ^ (string_of_int (l_count-1))
    ^ "]=" ^ display ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Add -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Sub -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Mul -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Div -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
```

```
| Pow -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| _ -> raise (Failure ("Matirx op can't apply on Float"))
)
|   (Float, Matrix) ->
(match op with
Mul -> let instruction = fst(expr_to_string_c expr) ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| _ -> raise (Failure ("Float can only apply Mul with Matrix
    "))
)
|   (Matrix, Float) -> raise (Failure ("Matirx op Float not
    allowed"))
|   (Matrix, Matrix) ->
(match op with
Mul -> let instruction = fst(expr_to_string_c expr) ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| DotMul -> let instruction = fst(expr_to_string_c expr) ^ "
    ;" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Add -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Sub -> let instruction = fst(expr_to_string_c expr) ^ ";"
    in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| _ -> raise (Failure ("Undefined Matrix operation"))
)
)
| Sum(a, u, e) -> let v, (v_set, m_set, l_count, c_count,
    i_list) = eval (v_set, m_set, l_count, c_count, i_list)
    expr in
let instruction = fst(expr_to_string_c expr) ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Prod(a, u, e) -> let v, (v_set, m_set, l_count, c_count,
    i_list) = eval (v_set, m_set, l_count, c_count, i_list)
    expr in
let instruction = fst(expr_to_string_c expr) ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
| Literal(f) -> (v_set, m_set, l_count, c_count, i_list)
| MLiteral(m) -> (v_set, m_set, l_count, c_count, i_list)
```

```
464 | Id(s) -> (v_set, m_set, l_count, c_count, i_list)
465 | Uniop(op, e) -> let v, (v_set, m_set, l_count, c_count,
        i_list) = eval (v_set, m_set, l_count, c_count, i_list) e
         in
466 (match v with
467 Matrix ->
468 ( match op with
469 Trans ->
470 let instruction = fst(expr_to_string_c expr) ^ ";" in
471 let i_list = instruction :: i_list in
472 (v_set, m_set, l_count, c_count, i_list)
473 | _ -> raise (Failure ("Un-supported matrix unit operator"))
474 )
475 | Float  ->
476 (
477 match op with
478 Sin ->  let instruction = fst(expr_to_string_c expr) ^ ";"
        in
479 let i_list = instruction :: i_list in
480 (v_set, m_set, l_count, c_count, i_list)
481 | Cos ->  let instruction = fst(expr_to_string_c expr) ^ ";"
         in
482 let i_list = instruction :: i_list in
483 (v_set, m_set, l_count, c_count, i_list)
484 | Log ->  let instruction = fst(expr_to_string_c expr) ^ ";"
         in
485 let i_list = instruction :: i_list in
486 (v_set, m_set, l_count, c_count, i_list)
487 | Tan ->  let instruction = fst(expr_to_string_c expr) ^ ";"
         in
488 let i_list = instruction :: i_list in
489 (v_set, m_set, l_count, c_count, i_list)
490 | _   ->  raise (Failure ("Un-supported float unit operator"
        ))
491 )
492 )
493 | Assign(vid, e) -> let v, (v_set, m_set, l_count, c_count,
        i_list) = eval (v_set, m_set, l_count, c_count, i_list) e
         in
494 (match v with
495 Float -> if StringSet.mem vid m_set then raise (Failure ("
        Cannot assign a new var with the same name of a matrix"))
         else
496 let v_set = StringSet.add vid v_set in
497 let instruction = fst(expr_to_string_c expr) ^ ";" in
498 let i_list = instruction :: i_list in
499 (v_set, m_set, l_count, c_count, i_list)
500 | Matrix -> if StringSet.mem vid v_set then raise (Failure (
        "Cannot assign a var with a matrix")) else
501 let m_set = StringSet.add vid m_set in
```

```
let instruction = fst(expr_to_string_c expr) ^ ";" in
let i_list = instruction :: i_list in
(v_set, m_set, l_count, c_count, i_list)
)
)
in

let process_stmt (v_set, m_set, l_count, c_count, i_list) (statement : statement) =
match statement with
Seq(expr_list) -> List.fold_left
(fun a b -> eval_top a b) (v_set, m_set, l_count, c_count, i_list) expr_list
in

(* Genereate variable set,
matrix set,
logic expression count,
formular evaluation count,
instruction list
*)
let (v_set, m_set, l_count, c_count, i_list) =
List.fold_left process_stmt (v_set, m_set, l_count, c_count, i_list) statements
in

let v_list = StringSet.elements(v_set) and m_list = StringSet.elements(m_set)

in

let fdef_latex_l =
List.rev (FPMap.fold (fun k v l -> String.escaped(formular_to_string v):: l) func_decls [])
in

let (v_str, c1) = List.fold_left
(fun (str, i) var -> (str ^ "double &" ^ var ^ " = vdata[" ^ string_of_int i ^ "];\n", i+1)) ("", 0) v_list
and (m_str, c2) = List.fold_left
(fun (str, i) var -> (str ^ "matrix &" ^ var ^ " = mdata[" ^ string_of_int i ^ "];\n", i+1)) ("", 0) m_list
in

(* Combine the strings *)
let code =
header v_list m_list l_count c_count fdef_latex_l title author date ^ "\n" ^

String.concat "\n\n" (List.rev f_decl_list) ^ "\n" ^
```

```
544 matrix_class ^ "\n" ^
545
546 "double vdata[NUM_OF_VARIABLES];\n" ^
547
548 "matrix mdata[NUM_OF_MATRIX_VARIABLES];\n" ^
549
550 v_str ^ "\n" ^
551 m_str ^ "\n" ^
552
553 dtos ^ "\n" ^
554 String.concat "\n\n" (List.rev f_list) ^ "\n" ^
555 main_preamble v_list m_list fdef_latex_l ^ "\n" ^
556
557 String.concat "\n" (List.map (fun ins -> "\t" ^ ins) (List.
       rev i_list)) ^ "\n" ^
558 latex_print lfile
559 in
560
561 (* Write string to file *)
562 let out  = open_out cppfile in
563 fprintf out "%s\n" code;
564 close_out out;
```

Listing 8.7: testall.sh

```
1  #!/bin/sh
2
3  EZMATH="./../../EZMath"
4
5  # Set time limit for all operations
6  ulimit -t 30
7
8  globallog=testall.log
9  rm -f $globallog
10 error=0
11 globalerror=0
12
13 keep=0
14
15 Usage() {
16     echo "Usage: testall.sh [options] [.tex files]"
17     echo "-k    Keep intermediate files"
18     echo "-h    Print this help"
19     exit 1
20 }
21
22 SignalError() {
23     if [ $error -eq 0 ] ; then
24       echo "FAILED"
25       error=1
```

```
26      fi
27
28      echo "  $1"
29  }
30
31  # Compare <outfile> <reffile> <difffile>
32  # Compares the outfile with reffile.  Differences, if any,
        written to difffile
33  Compare() {
34      echo diff -b $1 $2 ">" $3 1>&2
35      diff -b "$1" "$2" > "$3" 2>&1 || {
36          SignalError "$1 differs"
37          echo "FAILED $1 differs from $2" 1>&2
38      }
39  }
40
41  # Run <args>
42  # Report the command, run it, and report any errors
43  Run() {
44      echo $* 1>&2
45      eval $*
46  }
47
48
49  Check() {
50      error=0
51      basename=`echo $1 | sed 's/.*\\///
52                              s/.tex//'`
53      echo -n "$basename.tex..."
54      echo "###### Testing $basename" 1>&2
55
56      generatedfiles="output/interpret/${basename}.out"
57      referfile="ref/interpret/${basename}.out"
58      diff="output/interpret/${basename}.diff"
59      Run "$EZMATH"  "-i" $1 "1>" ${generatedfiles} "2>&1"
60      Compare ${generatedfiles} ${referfile} ${diff}
61
62      latexfile="output/compile/${basename}.tex"
63      referfile="ref/compile/${basename}.tex"
64      diff="output/compile/${basename}.diff"
65      Run "$EZMATH" "-c" $1 "-l" ${latexfile}
66      Run "g++ -std=c++11 result.cpp" &&
67      Run "./a.out" &&
68      Run "rm -f a.out result.cpp" &&
69      Compare ${latexfile} ${referfile} ${diff}
70
71      if [ $error -eq 0 ] ; then
72          if [ $keep -eq 0 ] ; then
73              rm -f $generatedfiles $latexfile
74          fi
```

```
75
76          echo "OK"
77          echo "###### SUCCESS" 1>&2
78      else
79          echo "###### FAILED" 1>&2
80          globalerror=$error
81      fi
82  }
83
84  while getopts kdpsh c; do
85      case $c in
86      k) # Keep intermediate files
87          keep=1
88          ;;
89      h) # Help
90          Usage
91          ;;
92      esac
93  done
94
95  shift `expr $OPTIND - 1`
96
97  if [ $# -ge 1 ]
98  then
99      files=$@
100 else
101     # files="tests/fail-*.tex tests/test-*.tex"
102     files="src/test-*.tex"
103 fi
104
105 for file in $files
106 do
107     case $file in
108     *test-*)
109         Check $file 2>> $globallog
110         ;;
111     *fail-*)
112         Check $file 2>> $globallog
113         ;;
114     *)
115         echo "unknown file type $file"
116         globalerror=1
117         ;;
118     esac
119 done
120
121 exit $globalerror
```

Listing 8.8: Makefile

```
OBJS = ast.cmo parser.cmo scanner.cmo header.cmo interpret.cmo \
  compile.cmo EZMath.cmo

TARFILES = Makefile testall.sh scanner.mll parser.mly \
  ast.ml compile.ml interpret.ml EZMath.ml header.ml

EZMath : $(OBJS)
  ocamlc -o EZMath str.cma unix.cma $(OBJS)

scanner.ml : scanner.mll
  ocamllex scanner.mll

parser.ml parser.mli : parser.mly
  ocamlyacc parser.mly

%.cmo : %.ml
  ocamlc -c $<

%.cmi : %.mli
  ocamlc -c $<

EZMath.tar.gz : $(TARFILES)
  cd .. && tar czf EZMath/EZMath.tar.gz $(TARFILES:%=EZMath/%)

.PHONY : clean
clean:
  rm -f EZMath parser.ml parser.mli scanner.ml tests.log \
  *.cmo *.cmi

# Generated by ocamldep *.ml *.mli
ast.cmo:
ast.cmx:
header.cmo: ast.cmo
header.cmx: ast.cmx
compile.cmo: header.cmo ast.cmo
compile.cmx: header.cmx ast.cmx
interpret.cmo: header.cmo ast.cmo
interpret.cmx: header.cmx ast.cmx
EZMath.cmo: scanner.cmo parser.cmi interpret.cmo compile.cmo header.cmo ast.cmo
EZMath.cmx: scanner.cmx parser.cmx interpret.cmx compile.cmx header.cmx ast.cmx
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmo
```