EZMath Final Report

Piaoyang Cui\textsuperscript{1}, Yi Wang\textsuperscript{2}, Shangjin Zhang\textsuperscript{3}, and Zhejiao Chen\textsuperscript{4}

\textsuperscript{1}pc2618@columbia.edu, Columbia University
\textsuperscript{2}yw2580@columbia.edu, Columbia University
\textsuperscript{3}sz2425@columbia.edu, Columbia University
\textsuperscript{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 \texttt{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). Furthermore, taking in the same source file paper.tex, compile it through the \texttt{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 \texttt{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 \texttt{EZMath} with well-known GCD (Greatest Common Divisor) algorithm. Suppose we have follow GCD program in gcd.tex file

\[
\text{gcd}(a,b) = \begin{cases}
    a & a == b \\
    \text{gcd}(a-b,b) & a > b \\
    \text{gcd}(a, b-a) & b > a
\end{cases}
\]

As you can see, we defined a formula called \texttt{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 \texttt{EZMath} only focus on math formula parts of gcd.tex and ignore texts.

Type \texttt{make} and let the OCaml generates \texttt{EZMath} program, there’re several options to get the final expected result (or summary).

One is interpreting:

\texttt{EZMath -i gcd.tex}
This command uses our inside interpreter program just as \texttt{microc} does, which will generate the output in the stdout:

\begin{verbatim}
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
\end{verbatim}

Our program takes the \texttt{gcd.tex}, creates a formula symbol table for \texttt{gcd}, then takes \texttt{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:

\texttt{EZMath \ -c gcd.tex}

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

\begin{verbatim}
...

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) {
  ...
\end{verbatim}
c_result[0]=string(""")+gcd(9, 21)=gcd("+dotos(9)+", "+dotos(21)+")="+dotos(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 \LaTeX\ compiler.

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!
Variables Definition

Matrix Definition

Formula Definition

\[
gcd(a, b) = \begin{cases} 
  a & a == b \\
  gcd(a - b, b) & a > b \\
  gcd(b, a - b) & b > a
\end{cases}
\]  \hspace{1cm} (1)

Logical Validation

Formula Evaluation

\[
gcd(9, 21) = gcd(9, 21) = 3
\]

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:
Note that, keyword pi will be replaced by 3.1415926..., e is recognized 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:

\[ c_1 = 3 \quad c_2 = 4 \quad c_3 = 5 \]
\[ c_1^2 + c_2^2 = c_3^2 \]

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:

\[ \text{sum} (p) = \sum_{i=1}^{p} i \]
\[ \text{prod} (p) = \prod_{i=1}^{p} i \]

\[ \text{sum} (100) \]
\[ \text{prod} (4) \]

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.

\[ \text{sum}_{} \quad \text{prod}_{} \]
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 \texttt{\LaTeX} matrix grammar.

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

Some examples:

\[
\begin{pmatrix}
10.5 & 20.2 & 30.5 \\
6.1 & 4 & 5 & 6 \\
7.3 & 7 & 8 & 9
\end{pmatrix}
\]

\[m = \begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}\]
3.1.3 Supported Matrix Operators

EZMath supports plus, minus, multiply, dot multiply and transpose (+, −, ×, ·, 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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>matrix add</td>
<td>A + B</td>
</tr>
<tr>
<td>-</td>
<td>matrix minus</td>
<td>A - B</td>
</tr>
<tr>
<td>\times or *</td>
<td>float/matrix multiply</td>
<td>A \times B</td>
</tr>
<tr>
<td>\cdot</td>
<td>multiply the corresponding elements</td>
<td>A \cdot B</td>
</tr>
<tr>
<td>↑{T}</td>
<td>matrix transpose</td>
<td>A\text{T}</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>The natural logarithm approximately equals to 2.71828.</td>
</tr>
<tr>
<td>( \pi )</td>
<td>( \pi ) approximately equal to 3.1415926.</td>
</tr>
<tr>
<td>\sin</td>
<td>Reserved for the ( \sin ) function</td>
</tr>
<tr>
<td>\cos</td>
<td>Reserved for the ( \cos ) function</td>
</tr>
<tr>
<td>\tan</td>
<td>Reserved for the ( \tan ) function</td>
</tr>
<tr>
<td>\log</td>
<td>Reserved for the ( \log ) function</td>
</tr>
</tbody>
</table>

3.5 Binary and Unit Operators

3.5.1 Float Number Operators

\texttt{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.

\texttt{EZMath} supports a subset of \LaTeX's original math symbols, including:

\( \sin, \cos, \tan, \log \)

3.5.2 Logical Operators

\texttt{EZMath} supports the following logical operators: larger, less than, equal, larger or equal, less or equal, inequal (\( \gt, \lt, =, \geq, \leq, \neq \)). Logical operators can be used in conditions of piecewise formulas and logical expressions.

3.6 Separators

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

<table>
<thead>
<tr>
<th>Separator</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(White space)</td>
<td>Separates tokens (see next section)</td>
</tr>
<tr>
<td>,</td>
<td>Separates expressions in statement</td>
</tr>
<tr>
<td>&amp;</td>
<td>Separates expression and condition in piecewise function</td>
</tr>
<tr>
<td>( \backslash )</td>
<td>Separates cases in piecewise function</td>
</tr>
</tbody>
</table>
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 \texttt{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 \texttt{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:

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

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 \( \sin() \).

\texttt{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.

\texttt{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:

\[
r(x) = 2 \times r(x/2)
\]

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

\texttt{EZMath} supports piecewise formulas. The usage of a piecewise formula is illustrated as follows:

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

As shown above, we use \\ to separate cases. And in a particular case, we use & to separate 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:

\begin{verbatim}
$$
\texttt{bar(x) = x + 1}
\texttt{foo(x) = bar(x) * 2}$n\end{verbatim}

3.9 Logical Expressions

Logical expressions are supported in $\LaTeX$ as following:

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

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

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

For the above expression, \texttt{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, \texttt{EZMath} compiler will check if \( a, b \) are defined and assigned values. If not, it will report error in \texttt{report.tex}. Otherwise, it will check if the inequality satisfies. If it does, it will report true in \texttt{report.tex}, false otherwise.

If the elements of an logical expression are all constants, \texttt{EZMath} will remember this logical expression and validate its correctness in \texttt{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 \texttt{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.

\[
\begin{align*}
\text{\$\$ a = 2, b = 3, c = 4\$\$} \\
\text{\$\$ a < 2, c > 3, b == \text{Sin}(a, b), pt(3,4,5)\$\$}
\end{align*}
\]

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

\[
\begin{align*}
\text{\$\$ (a + b), (\text{Sin}(2, 3) * g()) - 1\$\$}
\end{align*}
\]

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

\[
\begin{align*}
\text{\$\$ a = 2, b = 3 x = \text{Sin}(a, b)\$\$}
\end{align*}
\]

\( a = 2, b = 3 \) is the first statement and \( x = \text{Sin}(a, b) \) 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 \texttt{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 output 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, authors, 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 \texttt{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

<table>
<thead>
<tr>
<th>Space</th>
<th>Put space after punctuations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Write comment for each function group and important declaration. Nested comment not allowed in code.</td>
</tr>
<tr>
<td>Value Declaration</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.3 Project Timeline

![Overall Statistics](image)

Figure 4.1: Overall Statistics
Figure 4.2: GitHub Additions & Deletions

Figure 4.3: GitHub Commits
## 4.4 Roles and Responsibilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piaoyang Cui</td>
<td>Manager</td>
<td>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).</td>
</tr>
<tr>
<td>Yi Wang</td>
<td>Language Guru</td>
<td>Lead the design the language specs and features. Decide detail of language implementation. General contribution to project (design, code and test).</td>
</tr>
<tr>
<td>Shangjin Zhang</td>
<td>System Architect</td>
<td>Lead the design of interpreter and compiler architecture. Lead the prototyping of EZMath. General contribution to project (design, code and test).</td>
</tr>
<tr>
<td>Zhejiao Chen</td>
<td>Tester</td>
<td>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).</td>
</tr>
</tbody>
</table>

## 4.5 Development Environment

- OCaml Compiler: version 4.01.0
- C++ Compiler: GNU C+ 4.8, with C++11 support
- 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\(^1\).

![Architecture Diagram](image)

The **EZMath** compiler transforms an \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.

```
\begin{verbatim}
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
----------------------------------------
\end{verbatim}
```

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.

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

\[ f(a, b) = \begin{cases} 
3, & \text{if } x > 4. \text{ Or} \\
1, & \text{if } x > 4. 
\end{cases} \]

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'tests' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Add comment in compile.ml</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Shangjin_Zhang</td>
<td>More points</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Shangjin_Zhang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Shangjin_Zhang</td>
<td>add sum in interpreter</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a> into tests</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Modify testall script</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Add comment in parser</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Continue adding comment</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Add comment</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Add sectional comments</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Shangjin_Zhang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Shangjin_Zhang</td>
<td>implement sum as one type of expression</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Add Makefile</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Add negative sign</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Add more points</td>
</tr>
<tr>
<td>Date</td>
<td>Committer</td>
<td>Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Remove non-use file</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>fix issue#20</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>separate header and compile code generation hard code</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Add tan, log10, modify output cppfile</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Fix help message</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>remove test files</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Add ignore file, pass basic test</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' into uniop</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Fix issue no.19</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>modify interpret to print author title</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Yi_Wang</td>
<td>modify test paper, add {} in every power function</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Collecting ideas for slides</td>
</tr>
<tr>
<td>2014-12-15</td>
<td>Piaoyang_Cui</td>
<td>Change final _ case message</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>fix uniop in compile</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Superstar toplevel</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a> into uniop</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>exponent must be inside the BRACE pair</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>add sin cos log function in interpret</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Update readme status</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Improve format</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Improve interpreter output</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Zhejiao_Chen</td>
<td>add testall.sh</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Zhejiao_Chen</td>
<td>add testall.sh</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Ref in interpreter</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Shangjin_Zhang</td>
<td>separate function declaration and definition</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Apply float2str to all related files</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Shangjin_Zhang</td>
<td>change stringstream to dtos</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Add float2str for truncated float num</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Zhejiao_Chen</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
</tbody>
</table>
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 Shangjin_Zhang 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 ==)
<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Add matrix +/- in interpreter</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Zhejiao_Chen</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Zhejiao_Chen</td>
<td>add issue-1.tex: matrix +/- not supported</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>== should be higher than =. fix</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>add stdexcept</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>update</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Add platform test goal</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Print 1st version latex by C++</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Last step to fix the C++</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Add overloading to compiler</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Fix function redefinition</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Yi_Wang</td>
<td>reset scanner</td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-14</td>
<td>Piaoyang_Cui</td>
<td>Try to solve the overloading</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>add test cases</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>fix negative number issue</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>pretty print with formular definition in latex</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>add test cases</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Piaoyang_Cui</td>
<td>Unify all the raise_Failure message (Upcase, space, etc)</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>pretty print cpp file</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Piaoyang_Cui</td>
<td>Fix issue of parameter name not unique</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Piaoyang_Cui</td>
<td>add parameter test</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Zhejiao_Chen</td>
<td>add test cases</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Piaoyang_Cui</td>
<td>Update matrix exception message</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Piaoyang_Cui</td>
<td>Delete return -1.0</td>
</tr>
<tr>
<td>2014-12-13</td>
<td>Yi_Wang</td>
<td>Merge branch 'master' of <a href="https://github.com/i3wangyi/EZMath">https://github.com/i3wangyi/EZMath</a></td>
</tr>
</tbody>
</table>
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 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
github.com/i3wangyi/EZMath
2014-12-13 Piaoyang_Cui Improve toplevel: (1) Add version, help info and associative 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 construct 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.ml
2014-12-05 Shangjin_Zhang Change space to any space in scanner.ml
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 examples
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  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, prohibited 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 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  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
   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  Testing conflict resolving 2
2014-11-16  Yi_Wang  Testing conflict resolving
   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
   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
   com/i3wangyi/EZMath
8.2 Source Code

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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZMath.ml</td>
<td>82</td>
</tr>
<tr>
<td>header.ml</td>
<td>411</td>
</tr>
<tr>
<td>interpret.ml</td>
<td>378</td>
</tr>
<tr>
<td>compile.ml</td>
<td>564</td>
</tr>
<tr>
<td>scanner.mll</td>
<td>72</td>
</tr>
<tr>
<td>parser.mly</td>
<td>143</td>
</tr>
<tr>
<td>ast.ml</td>
<td>98</td>
</tr>
<tr>
<td>Makefile</td>
<td>45</td>
</tr>
<tr>
<td>testall.sh</td>
<td>121</td>
</tr>
</tbody>
</table>
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

;;
(* Matrix printing utilities *)

let matrix_s m = " [" ^ String . concat , " ( Array . to_list ( Array . map ( fun e -> "( " ^ ( float2str e) ^ " )" ) m)) ^ " ]"

let traverse_matrix any = Array . to_list ( Array . map ( fun e -> "(" ^ matrix_s e ^ ")" ) any )

let matrix_decl_s m = "{\n " ^ String . concat , " \n " (traverse_matrix m) ^ 
}"

(* Expression/Statement printing utilities *)

let rec expr_s = function
  | Literal (l) -> " Literal " ^ float2str l
  | MLiteral (m) -> " MLiteral " ^ matrix_decl_s m
  | Id(s) -> "Id " ^ s
  | Binop (e1 , o, e2) -> " Binop (" ^ expr_s e1 ^ ") " ^ ( match o with Add -> " Add " | Sub -> " Sub " | Mul -> " Mul " | Div -> " Div " | Equal -> " Equal " | Neq -> " Neq " | Less -> " Less " | Leq -> " Leq " | Greater -> " Greater " | Geq -> " Geq " | Pow -> " Pow " | DotMul -> " DotMul " ) ^ " (" ^ expr_s e2 ^ ")"
  | Sum (down , up , e) -> " Sum (" ^ ( expr_s down ) ^ ") " ^ " (" ^ ( expr_s up ) ^ ") " ^ " (" ^ expr_s e ^ ")"
  | Prod (down , up , e) -> " Prod (" ^ ( expr_s down ) ^ ") " ^ " (" ^ ( expr_s up ) ^ ") " ^ " (" ^ expr_s e ^ ")"
  | Uniop (o, e) -> " Uniop (" ^ ( match o with Sin -> " Sin " | Trans -> " Trans " | Cos -> " Cos " | Log -> " Log " | Tan -> " Tan " | Tanh -> " Tanh" ) ^ " (" ^ expr_s e ^ ")"
  | Assign (v, e) -> " Assign " ^ v ^ " (" ^ expr_s e ^ ")"
  | Call (f, es) -> " Call " ^ f ^ " [" ^ ( List . map ( fun e -> "(" ^ expr_s e ^ ")" ) es) ^ "] "

let traverse any = List . map ( fun e -> "(" ^ expr_s e ^ ")" ) any

let stmt_s = function
  | Seq (ss) -> " Seq " ^ ( String . concat , " \n " (traverse ss) ^ " ]"

(* Function printing utilities *)

let case_s c = "\n { expression = \"" ^ ( expr_s c.expression) ^ 
 condition = [" ^ ( expr_s c.condition) ^ " ] \n"
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 ",
(List.map func_decl_s funcs) ^ "],\n[ " ^ String.concat "\n(List.map stmt_s stats) ^ "])"

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
| "\"\$\" { code lexbuf }
| "\"%\" { comment_in_code lexbuf}
| "\"\" \" \" { TRANS }
| "\"\" \" { FLOAT(3.1415926) }
| "\"\" { FLOAT(2.71828) }
| (ident as fid) [ ' ' 't' 'r' 'n' ]* '(',') { FID(fid) }
| '_.' { UNDERSCORE }
| '(' { LPAREN }
| ')' { RPAREN }
| '{' { LBRACE }
| '}' { RBRACE }
| ',' { COMMA }
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }

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 */

/* Float Operation */

/* Matrix Binop */

/* Precedence and Association */

%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 to 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 { Sum($4, $8, $11) }
| PROD UNDERSCORE LBRACE expr RBRACE CARET LBRACE expr RBRACE { Prod($4, $8, $11) }
| VID ASSIGN expr { Assign($1, $3) }
| FID arguments RPAREN { Call($1, $2) }
| LPAREN arguments 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 ]
Listing 8.4: header.ml

```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" 
```

51
const string Title = "\\" title "\\";\\n"

const string Author = "\\" author "\\";\\n"

const string Date = "\\" date "\\";\\n"

let matrix_class = "

class matrix
{
  // declare a vector of vectors of type double
  vector<vector<double>> s;

public:
  int m, n;

  // Initialize the size of s to row by col
  matrix(int row = 1, int col = 1): m(row), n(col), s(row, vector<double>(col)) {};
  matrix(int row, int col, const double data[]): m(row), n(col), s(row, vector<double>(col)) {
  for(int i = 0; i < row; i++)
    for(int j = 0; j < col; j++)
      s[i][j] = data[i * col + j];
  }

  string printm();

  // declare the operators +,-,*~,DotMul as friends and with return type matrix
  friend matrix operator+(const matrix&, const matrix&);
  friend matrix operator-(const matrix&, const matrix&);
  friend matrix operator*(const matrix&, const matrix&);
  friend matrix operator*(const matrix&, double);
  friend matrix operator*(double, const matrix&);
  friend matrix operator~(const matrix&);
  friend matrix DotMul(const matrix&, const matrix&);
};

string matrix::printm()
stringstream lex;
lex << "\\begin { bmatrix }" << endl;
for (int i = 0; i < m; i++)
{
for (int j = 0; j < n; j++)
{
lex << this->s[i][j];
lex << ((j < n - 1) ? " & " : ((i < m - 1)? " \\
: ") );
}
lex << endl;
}
lex << "\\end { bmatrix }";
return lex.str();
}

class matrix
{
matrix operator+(const matrix& a, const matrix& b)
{
//declare a matrix temp to store the result and return this
matrixttemp(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;
}
matrix operator-(const matrix& a, const matrix& b)
{
if (a.m != b.m || a.n != b.n)
throw std::runtime_error("Invalid matrix subtraction:
  dimensions not match\n");
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;
}
matrix operator*(const matrix& a, const matrix& b)
{
if (a.n != b.m)
throw std::runtime_error("Invalid matrix multiplication:
  dimensions not match\n");
matrix temp(a.m, b.n);
for (int i = 0; i < a.m; i++)
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)"
let main_preamble v_list m_list fdef_latex_l =
"/**
* Main Function
**/
int main(int argc, char ** argv) {
"^"tstring var_def [NUM_OF_VARIABLES] = {{"^
" (List.map (fun var -> "\"" ^ var ^ "\"\") v_list) ^ "};\n
"^"tstring matrix_def [NUM_OF_MATRIX_VARIABLES] = {{"^
" (List.map (fun var -> "\"" ^ var ^ "\"\") m_list) ^ "};\n
"^"tstring formular_def [NUM_OF_FORMULAR_DEFINITION] = {{"^
" (List.map (fun f -> "\"" ^ f ^ "\"\") fdef_latex_l) ^ "};\n
"^"tstring l_result [NUM_OF_LOGICAL_VALIDATION];\n
"^"tstring c_result [NUM_OF_FORMULAR_EVALUATION];\n
let latex_print file =
"^"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";


for (int i = 0; i < NUM_OF_VARIABLES; i++) {
    file << "\\section*{Variables Definition}\n";
    file << "\\begin{\begin{array}{c} \n" << var_def[i] << " = " << vdata[i] << " \n\n";
}

for (int i = 0; i < NUM_OF_MATRIX_VARIABLES; i++) {
    file << "\\section*{Matrix Definition}\n";
    file << "\\begin{array}{c} \n" << matrix_def[i] << " = " << mdata[i].printm() << " \n\n";
}

file << "\\section*{Formula Definition}\n";
file << "\\begin{\begin{array}{c} \n";
for (int i = 0; i < NUM_OF_FORMULAR_DEFINITION; i++) {
    if (i != NUM_OF_FORMULAR_DEFINITION - 1) {
        file << formular_def[i] << "\\\\\n";
    } else {
        file << formular_def[i] << "\n";
    }
}
file << "\\end{\begin{array}{c} \n";

file << "\\section*{Logical Validation}\n";
for (int i = 0; i < NUM_OF_LOGICAL_VALIDATION; i++) {
    file << "\\section*{Logical Validation}\n";
    file << "\\begin{\begin{array}{c} \n" << l_result[i] << " \n\n";
}

file << "\\section*{Formula Evaluation}\n";
for (int i = 0; i < NUM_OF_FORMULAR_EVALUATION; i++) {
    file << "\\section*{Formula Evaluation}\n";
    file << "\\begin{\begin{array}{c} \n" << c_result[i] << " \n\n";
}

file << "\\end{document}\n";
file.close();
return 0;
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" (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
| Sub  -> let (op_s, i) = "-", 3 in ((if i1 >= i then s1 else
  "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
  s2 ^ ")")), i |
| Mul -> let (op_s, i) = ".", 4 in ((if i1 >= i then s1 else
  "(" ^ s1 ^ ")") ^ op_s ^ (if i2 >= i then s2 else "(" ^
  s2 ^ ")")), i |
| 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 |
| DotMul -> let i = 5 in (" DotMul(" ^ s1 ^ ", " ^ s2 ^ ")"), i |

Sum (a, u, e) -> let vid, ex = match a with
Assign (vid, ex) -> vid, ex
_ -> raise (Failure ("first parameter in sum should be
assign"))

in
[] (int bottom, int top) {double sum=0; for (int "; vid "; =
bottom; "; vid "; <top+1; "; vid "; ++) sum+=` (fst( 
expr_to_string_c e))
`; return sum;}` (int)(" (fst(expr_to_string_c ex))
`; "}, ` (int)(" (fst(expr_to_string_c u)) ` "))", 6
Prob (a, u, e) -> let vid, ex = match a with
Assign (vid, ex) -> vid, ex
_ -> raise (Failure ("first parameter in prob should be
assign"))

in
[] (int bottom, int top) {double prod=1; for (int "; vid "; =
bottom; "; vid "; <top+1; "; vid "; ++) prod*=` (fst( 
expr_to_string_c e))
let expr_list_to_string_p (seperator : string) (para_list : expr list) : string = String.concat seperater (List.map (fun e -> "dtos (" ^ (fst (expr_to_string_c e)) ^ ")") para_list)

let expr_list_to_string (expr : expr list) : string = String.concat ", " (List.map (fun e -> (fst (expr_to_string e))) expr)

and expr_to_string (expr : expr) : string * int = match expr with
| Literal(f) -> (float2str f), 0
| Id(s) -> s, 0
| Assign(vid, e) -> (vid ^ " = " ^ (fst (expr_to_string_c e)))

match op with
| Trans -> " - " ^ (" ^ (fst (expr_to_string_c e)) ^ ")", 6
| Sin -> " sin " ^ (" ^ (fst (expr_to_string_c e)) ^ ")", 6
| Cos -> " cos " ^ (" ^ (fst (expr_to_string_c e)) ^ ")", 6
| Log -> " log10 " ^ (" ^ (fst (expr_to_string_c e)) ^ ")", 6
| Tan -> " tan " ^ (" ^ (fst (expr_to_string_c e)) ^ ")", 6
let case_to_string (case : case) : string =
  let expression = fst (expr_to_string case.expression) in
  match op with
  | Sin -> "\sin {\{ (fst (expr_to_string a)) \} ^}"
  | Cos -> "\cos {\{ (fst (expr_to_string a)) \} ^}"
  | Log -> "\log_{10} {\{ (fst (expr_to_string a)) \} ^}"
  | Tan -> "\tan {\{ (fst (expr_to_string a)) \} ^}"
  | _ -> raise (Failure ("No matrix in logical expressions and function calls"))
 );

let expression = fst (expr_to_string 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}\n" in
  fname ^ "(" ^ parameter ^ ") = " ^ definition
;;

Listing 8.5: interpret.ml

open Ast
open Header

module StringMap = Map.Make(String)
module StringSet = Set.Make(String)
(* User-defined Map, key: (string * int) For Formular Overloading *)

module FunctionParamLen =
struct
    type t = string * int
    let compare (x1, x2) (y1, y2) =
        if x1 < y1 then -1
        else if x1 > y1 then 1
        else if x2 < y2 then -1
        else if x2 > y2 then 1
        else 0
end ;;

module FPMap = Map.Make(FunctionParamLen)

(* Symbol table: Information about all the names in scope *)
type v_table = float StringMap.t

type f_table = formular FPMap.t

type m_table = matrix StringMap.t

type global_config = v_table * m_table

type rettype = Float of float | Matrix of matrix

(* Global mutable list, information about formualr evaluation instruction and logical evaluation instruction *)
let fcall_strs = ref [];;
let lvali_strs = ref [];;

(* Get item in tripe-pairs *)
let first (x,_,_) = x;;
let second (_,y,_) = y;;
let third (_,_,z) = z;;

(* Matrix Multiplication *)
let matrix_constant_mul (lambda : float) (x : matrix) : matrix =
    let x_row = Array.length x in
    if x_row = 0 then raise (Failure "Invalid matrix with row# = 0") else let x_col = Array.length x.(0) in
    if x_col = 0 then raise (Failure "Invalid matrix with col# = 0") else
let z = Array.make_matrix x_row x_col 0. in
for i = 0 to x_row-1 do
for j = 0 to x_col-1 do
z.(i).(j) <- lambda *. x.(i).(j)
done
done;
(* Matrix Multiplication *)
let matrix_multiply (x: matrix) (y: matrix) : matrix =
let x_row = Array.length x and y_row = Array.length y in
if x_row = 0 then raise (Failure("Invalid matrix with row # = 0")) else let x_col = Array.length x.(0) in
if y_row = 0 then raise (Failure("Invalid matrix with row # = 0")) else let y_col = Array.length y.(0) in
if (x_col = 0) || (y_col = 0) then raise (Failure("Invalid matrix with col # = 0")) else
if x_col <> y_row then raise (Failure("Invalid matrix multiply ")) else
let z = Array.make_matrix x_row y_col 0. in
for i = 0 to x_row -1 do
for j = 0 to y_col -1 do
for k = 0 to y_row -1 do
z.(i).(j) <- z.(i).(j) +. x.(i).(k) *. y.(k).(j)
done
done;
(* Matrix Dot Multiplication *)
let matrix_dot_multiply (x: matrix) (y: matrix) : matrix =
let x_row = Array.length x and y_row = Array.length y in
if x_row = 0 then raise (Failure("Invalid matrix with row # = 0")) else let x_col = Array.length x.(0) in
if y_row = 0 then raise (Failure("Invalid matrix with row # = 0")) else let y_col = Array.length y.(0) in
if (x_col = 0) || (y_col = 0) then raise (Failure("Invalid matrix with col # = 0")) else
if (x_row <> y_row) || (x_col <> y_col) then raise (Failure("Invalid matrix dot multiply ")) else
let z = Array.make_matrix x_row x_col 0. in
for i = 0 to x_row-1 do
for j = 0 to x_col-1 do
z.(i).(j) <- x.(i).(j) *. y.(i).(j)
done
done;
(* Matrix Addition *)
let matrix_add (x: matrix) (y: matrix) : matrix =
let x_row = Array.length x and y_row = Array.length y in
if x_row = 0 then raise (Failure ("Invalid matrix with row # = 0")) else let x_col = Array.length x.(0) in
if y_row = 0 then raise (Failure ("Invalid matrix with row # = 0")) else let y_col = Array.length y.(0) in
if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid matrix with col# = 0")) else
if (x_row <> y_row) || (x_col <> y_col) then raise (Failure ("Invalid matrix dot multiply ")) else
let z = Array.make_matrix x_row x_col 0. in
for i = 0 to x_row -1 do
  for j = 0 to x_col -1 do
    z.(i).(j) <- x.(i).(j) +. y.(i).(j)
done
done;
(z)

(* Matrix subtraction *)
let matrix_sub (x: matrix) (y: matrix) : matrix =
let x_row = Array.length x and y_row = Array.length y in
if x_row = 0 then raise (Failure ("Invalid matrix with row # = 0")) else let x_col = Array.length x.(0) in
if y_row = 0 then raise (Failure ("Invalid matrix with row # = 0")) else let y_col = Array.length y.(0) in
if (x_col = 0) || (y_col = 0) then raise (Failure ("Invalid matrix with col# = 0")) else
if (x_row <> y_row) || (x_col <> y_col) then raise (Failure ("Invalid matrix dot multiply ")) else
let z = Array.make_matrix x_row x_col 0. in
for i = 0 to x_row -1 do
  for j = 0 to x_col -1 do
    z.(i).(j) <- x.(i).(j) -. y.(i).(j)
done
done;
(z)

(* Matrix Transpose *)
let matrix_transpose (x: matrix) : matrix =
let x_row = Array.length x in
if x_row = 0 then raise (Failure ("Invalid matrix with row # = 0")) else let x_col = Array.length x.(0) in
if x_col = 0 then raise (Failure ("Invalid matrix with col# = 0")) else
let z = Array.make_matrix x_col x_row 0. in
for i = 0 to x_col -1 do
  for j = 0 to x_row -1 do
    z.(i).(j) <- x.(j).(i)
done
done;
(z)
Executing program, output result in stdout

@parameter : program : formulars * statements (defined in ast.ml)

@parameter : (title, author)

*)

let run ((formulars, statements) : program) (title, author)
 : unit =

(* Build a symbol table for function declarations *)

let func_decls : f_table = List.fold_left

(* Semantics checking: parameter names should be unique;
 function with same signature (name + # of parameters)
 should be unique *)

(f fun functions func -> ignore(List.fold_left (fun param_map
 expr -> match expr with
 Id(n) -> if StringSet.mem n param_map then raise (Failure ("Parameter redefinition of " ^ n)) else
 (StringSet.add n param_map)
 | _ -> raise (Failure ("Parameter is not an ID")))
 )
 StringSet.empty func.parameter);

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."))
 else FPMap.add (func.fname, List.length func.parameter) func
 functions)

FPMap.empty formulars

in

let func_name = FPMap.fold (fun k v set -> StringSet.add (
 fst(k)) set) func_decls StringSet.empty
in

(* Initial Empty Matrix table, id(string) -> val(matrix) *)

let global_matrices : m_table = StringMap.empty
in

(* Initial Empty Variable table, id(string) -> val(Float) *)

let global_vars : v_table = StringMap.empty
in

(* Formular execution utilities *)

let rec exec_formular (fdecl : formular)(actuals : float
 list)(global_vars : v_table) : rettype * v_table =

let get_name (name:expr) = match name with
Id(n) -> n
let find_case (config : rettype * v_table * bool) (case : case) : bool = let global_vars = second(config) in let v, (_, _) = eval (global_vars, global_matrices) case. condition in match v with Float(f) -> (f > 0.) | _ -> raise (Failure ("Matrix is not allowed in case condition")) in 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 in match fdecl.definition with Regular(e) -> let (result, (local_vars, global_matrices)) = eval (local_vars, global_matrices) e in (result, global_vars) (* Note: global_matrix is empty *) | Piecewise(case_list) -> let (result, local_vars, is_found) = List.fold_left (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)) (Float(0.), local_vars, false) case_list in if is_found then (result, global_vars) else raise (Failure ("Piecewise not complete")) and (* Main evaluation function *) eval ((global_vars : v_table), (global_matrices : m_table)) (exp : expr) : rettype * global_config = match exp with (* Return Constant *) Literal(f) -> Float(f), (global_vars, global_matrices) | MLiteral(m) -> Matrix(m), (global_vars, global_matrices) (* first try to find in float vars, then try to find in matrix vars *) | Id(s) -> if StringMap.mem s global_vars then Float((StringMap.find s global_vars)), (global_vars, global_matrices) else if StringMap.mem s global_matrices then Matrix((StringMap.find s global_matrices)), (global_vars, global_matrices)
else raise (Failure ("Undeclared identifier " ^ s))
| Assign(vid, e) ->
if StringSet.mem vid func_name then raise (Failure ("Cannot assign a new var with the same name of a formular"))
else
let (v, (global_vars, global_matrices)) = eval (global_vars, global_matrices) e in
(match v with
Float(f) -> if StringMap.mem vid global_matrices then raise (Failure ("Cannot assign a new var with the same name of a matrix")) else
(Float(f), (StringMap.add vid f global_vars, global_matrices))
| Matrix(m) -> if StringMap.mem vid global_vars then raise (Failure ("Cannot assign a var with a matrix")) else
Matrix(m), (global_vars, (StringMap.add vid m global_matrices))

| Sum(a, u, e) -> let vid, ex = match a with
Assign(vid, ex) -> vid, ex
| _ -> raise (Failure ("first parameter in sum should be assign"))
in
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 sum = ref 0. in
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 sum"))
in
sum := !sum +. ret
done;
Float(!sum), (global_vars, global_matrices)
| _ -> raise (Failure ("no matrix in sum"))
)
| Prod(a, u, e) -> let vid, ex = match a with
Assign(vid, ex) -> vid, ex
| _ -> raise (Failure ("first parameter in prod should be assign"))
in
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) -> (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 uni 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)


68
Div -> if f2 = 0. then raise (Failure ("Division by 0!"))
else Float(f1 / f2)
Equal -> Float(boolean (f1 = f2))
Neq -> Float(boolean (f1 <> f2))
Less -> Float(boolean (f1 < f2))
Leq -> Float(boolean (f1 <= f2))
Greater -> Float(boolean (f1 > f2))
Geq -> Float(boolean (f1 >= f2))
Pow -> Float(f1 ** f2)
_ -> raise (Failure ("Matrix op can’t apply on float")), (global_vars, global_matrices)
Mul -> (Matrix(matrix_constant_mul f1 m1)), (global_vars, global_matrices)
_ -> raise (Failure ("Float can only apply Mul with matrix "))
(Matrix(m1), Float(f1)) -> raise (Failure ("Matrix op Float not allowed"))
(Mul -> (Matrix(matrix_multiply m1 m2)), (global_vars, global_matrices)) (* Matrix Multiplication operation*)
DotMul -> (Matrix(matrix_dot_multiply m1 m2)), (global_vars, global_matrices)) (* Matrix DotMul operation*)
Add -> (Matrix(matrix_add m1 m2)), (global_vars, global_matrices)) (* Matrix add operation*)
Sub -> (Matrix(matrix_sub m1 m2)), (global_vars, global_matrices)) (* Matrix sub operation*)
_ -> raise (Failure ("Matrix op Matrix not allowed"))
Call(f, para_list) ->
let fdecl =
try
FPMap . find (f, List . length para_list) func_decls
with Not_found -> raise (Failure ("Undefined function " ^ f) in
let (actuals: float list), (global_vars, global_matrices) = 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"))
69
let (r, v) = exec_formular fdecl actuals global_vars
in let f = match r with Matrix(m) -> raise (Failure ("Matrix not allowed in function return type")) | Float(f) -> f
in (Float(f), (v, global_matrices))

(* 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 (el, 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
  )
in
let process_stmt (global_config : global_config) (statement : statement) =
  match statement with
  Seq(expr_list) -> (List.fold_left
    (fun a b -> snd (eval_top a b)) global_config expr_list)
in
(* Final Report Printing *)
let _ = print_endline (title ^ " " ^ author) in
let _ = print_endline "
Formular Definitions\n------------------------------\n"
in
let _ = List.iter (fun f -> print_endline(
  formular_to_string_interpreter f)) formulars in
let (global_vars, global_matrices) = List.fold_left
  process_stmt (global_vars, global_matrices) statements in
let _ = print_endline "
Formular Evaluation\n------------------------------\n"
in
let _ = List.iter print_endline (List.rev ! fcall_strs) in
let _ = List.iter print_endline (List.rev ! lvali_strs) in

StringMap.iter (fun key value -> print_endline(key ^ " = " ^
  float2str(value))) global_vars;
print_endline "------------------------------\n"

StringMap.iter (fun key value -> print_endline(key ^ " = " ^
  Ast.matrix_decl_s(value))) global_matrices

Listing 8.6: compile.ml

open Header
open Ast
open Printf (* Header for write file *)
module StringSet = Set.Make(String)
module StringMap = Map.Make(String)
(* User-defined Map, key: (string * int) * For Formular Overloading *)

module FunctionParamLen =
struct
  type t = string * int
  let compare (x1,x2) (y1,y2) =
    if x1 < y1 then -1
    else if x1 > y1 then 1
    else if x2 < y2 then -1
    else if x2 > y2 then 1
    else 0
end ;;

module FPMap = Map.Make(FunctionParamLen)

(* Symbol table: Information about all the names in scope *)

type f_table = formular FPMap.t

type v_table = string list StringMap.t

type rettype = Float | Matrix

(* Translate program into C++ code @parameter : program : formulars * statements (defined in ast.ml) @parameter : (title, author, cpp file name, latex file name) *)

let translate ((formulars, statements) : program) (title, author, cppfile, lfile): unit =

(* global float variable table *)
let v_set = StringSet.empty in

(* global matrix variable table *)
let m_set = StringSet.empty in

(* # of logical expressions *)
let l_count = 0 in

(* # of function calls *)
let c_count = 0 in

(* List of instructions in C++ *)
let i_list = [] in
let f_list = [] in

let f_decl_list = [] in

let func_vars : v_table = StringMap.empty in

let func_decls : f_table = List.fold_left (fun functions func -> ignore(List.fold_left (fun param_map expr -> match expr with
    Id(n) -> if StringMap.mem n param_map then raise (Failure("Parameter redefinition of " ^ n)) else
        (StringMap.add n 0 param_map)
    | _ -> raise (Failure("Parameter is not an ID")))
StringMap.empty func.parameter);
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.
))
else FPMap.add (func.fname, List.length func.parameter) func functions)
FPMap.empty formulars

let func_name = FPMap.fold (fun k v set -> StringSet.add (fst(k)) set) func_decls StringSet.empty in

let rec formular_eval (v_set, g_set) (expr : expr) =
    match expr with
    Call(f, para_list) -> let fdecl = try
        FPMap.find (f, List.length para_list) func_decls
    with Not_found -> raise (Failure("Undefined function " ^ f))
    in
    if (List.length fdecl.parameter)<=>(List.length para_list)
    then raise (Failure("Unmatched parameters"))
    else let (v_set, g_set) = List.fold_left
        (fun (v_set, g_set) actual ->
        let v, (v_set, g_set) = formular_eval (v_set, g_set) actual
        in
        match v with
        |Id(n) -> if StringMap.mem n param_map then raise (Failure("Parameter redefinition of " ^ n)) else
            (StringMap.add n 0 param_map)
        | _ -> raise (Failure("Parameter is not an ID")))
        StringMap.empty func.parameter);
    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.
))
    else FPMap.add (func.fname, List.length func.parameter) func functions)
    FPMap.empty formulars

in

Float -> (v_set, g_set)
| Matrix -> raise (Failure("Formular can not take in matrix\n"))
| (v_set, g_set) (List.rev para_list) in
Float, (v_set, g_set)
| Binop(e1, op, e2) -> let v1, (v_set, g_set) = 
formular_eval (v_set, g_set) e1 in 
let v2, (v_set, g_set) = formular_eval (v_set, g_set) e2 in
(match (v1,v2) with
| (Float, Float) ->
(match op with
| Equal -> Float, (v_set, g_set)
| Neq -> Float, (v_set, g_set)
| Less -> Float, (v_set, g_set)
| Leq -> Float, (v_set, g_set)
| Greater -> Float, (v_set, g_set)
| Geq -> Float, (v_set, g_set)
| Add -> Float, (v_set, g_set)
| Sub -> Float, (v_set, g_set)
| Mul -> Float, (v_set, g_set)
| Div -> Float, (v_set, g_set)
| Pow -> Float, (v_set, g_set)
| _ -> raise (Failure("Matrix op can’t apply on Float"))
) )
| _ -> raise (Failure("No matrix in formular"))
) )
| Sum(a, u, e) -> let vid, ex = match a with
| Assign (vid, ex) -> vid, ex
| _ -> raise (Failure("first parameter in sum 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 v_set else
StringSet.remove vid tmp_v_set in
Float, (v_set, g_set)
| _ -> raise (Failure("no matrix in sum"))
| Prod(a, u, e) -> let vid, ex = match a with
| Assign(vid, ex) -> vid, ex
| _ -> raise (Failure("first parameter in prod should be assign"))
)
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 ("Unsupported 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"))
    )
  | _ -> raise (Failure ("Un - supported float unit operator"))
  | _ -> raise (Failure ("No matrix in formular"))

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"))

let local_set = List.fold_left
  (fun set expr -> StringSet.add (get_name expr) set)
  StringSet.empty fdecl.parameter
  in
let process_case (v_set, g_set) (case : case) = let v, (v_set, g_set) = formular_eval (v_set, g_set) case.
    expression in
let v, (v_set, g_set) = formular_eval (v_set, g_set) case.
    condition in
let ret = "if(" ^ fst (expr_to_string_c case.condition) ^ ")
    return "
    ^ fst (expr_to_string_c case.expression) ^ ";" in
(v_set, g_set), ret

match fdecl.definition with
| Regular(e) -> let v, (v_set, g_set) = formular_eval (local_set, StringSet.empty) e in
    let func_vars = StringMap.add fdecl.fname (StringSet.
        elements g_set) func_vars in
    let decl = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        (List.map (fun e -> "double " ^ fst (expr_to_string_c e)
            ) fdecl.parameter) ^ ");" in
    let ret = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        (List.map (fun e -> "double " ^ fst (expr_to_string_c e)
            ) fdecl.parameter) ^ "){" in
    ^ "return " ^ fst (expr_to_string_c e) ^ ";\n}\n"
    func_vars, ret::f_list, decl::f_decl_list

| Piecewise(case_list) -> let (v_set, g_set), tmp = List.
    fold_left (fun ((v_set, g_set), ret) case ->
        let (v_set, g_set), r = process_case (v_set, g_set) case in
        (v_set, g_set), ret ^ "\n\t" ^ r)
    (local_set, StringSet.empty), "") case_list
in
    let func_vars = StringMap.add fdecl.fname (StringSet.
        elements g_set) func_vars in
    let decl = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        (List.map (fun e -> "double " ^ fst (expr_to_string_c e)
            ) fdecl.parameter) ^ ");" in
    let ret = "double " ^ fdecl.fname ^ "(" ^ String.concat ",
        (List.map (fun e -> "double " ^ fst (expr_to_string_c e)
            ) fdecl.parameter) ^ "){" in
    ^ tmp ^ "\n\t\ntoow std::runtime_error("Illegal parameter in
piecwise function " ^ fdecl.fname ^ ");\n}\n"
    func_vars, ret::f_list, decl::f_decl_list

let (func_vars, f_list, f_decl_list) = List.fold_left process_formular (func_vars, f_list, 
    f_decl_list) formulars
let rec eval (v_set, m_set, l_count, c_count, i_list) (expr : expr) =
(match expr with
  Call (f, para_list) -> let fdecl =
    try FMap.find (f, List.length para_list) func_decls
    with Not_found -> raise (Failure ("Undefined function " ^ f))
    in
    let g_list = StringMap.find f func_vars
    in
    List.iter (fun g -> if StringSet.mem g v_set then () else
        raise (Failure ("Uninitialized global variable " ^ g))
    )
    g_list;
    if (List.length fdecl.parameter)<>(List.length para_list)
    then raise (Failure ("Unmatched parameters"))
    else
    let (v_set, m_set, l_count, c_count, i_list) = List.fold_left
        (fun (v_set, m_set, l_count, c_count, i_list) actual ->
            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
            match v with
            Float -> (v_set, m_set, l_count, c_count, i_list)
            | Matrix -> raise (Failure ("Formular can not take in matrix "))
        ) (v_set, m_set, l_count, c_count, i_list) (List.rev para_list)
    in
    (v_set, m_set, l_count, c_count, i_list)
  | 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
    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
    (match (v1, v2) with
    Equal -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Neq -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Less -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Leq -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Greater -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Geq -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Add -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Sub -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Mul -> Float, (v_set, m_set, l_count, c_count, i_list)
    | Div -> Float, (v_set, m_set, l_count, c_count, i_list)
| Pow -> Float, (v_set, m_set, l_count, c_count, i_list) |
| _ -> raise (Failure ("Matrix op can’t apply on Float")) |
| (Float, Matrix) -> |
| (match op with |
| Mul -> Matrix, (v_set, m_set, l_count, c_count, i_list) |
| _ -> raise (Failure ("Float can only apply Mul with Matrix")) |
| ) |
| (Matrix, Float) -> raise (Failure ("Matrix op Float not allowed")) |
| (Matrix, Matrix) -> |
| (match op with |
| Mul -> Matrix, (v_set, m_set, l_count, c_count, i_list) |
| DotMul -> Matrix, (v_set, m_set, l_count, c_count, i_list) |
| Add -> Matrix, (v_set, m_set, l_count, c_count, i_list) |
| Sub -> Matrix, (v_set, m_set, l_count, c_count, i_list) |
| _ -> raise (Failure ("Undefined Matrix operation")) |
| ) |
| Sum(a, u, e) -> let vid, ex = match a with |
| Assign (vid, ex) -> vid, ex |
| _ -> raise (Failure ("first parameter in sum should be assign")) |
| in |
| 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 |
| 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 |
| 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, m_set, l_count, c_count, i_list) = eval (tmp_v_set, m_set, l_count, c_count, i_list) e in |
| (match (v1, v2, v3) with |
| (Float, Float, Float) -> let v_set = if StringSet.mem vid v_set then v_set else StringSet.remove vid tmp_v_set in |
| Float, (v_set, m_set, l_count, c_count, i_list) |
| _ -> raise (Failure ("no matrix in sum")) |
| ) |
| Prod(a, u, e) -> let vid, ex = match a with |
| Assign (vid, ex) -> vid, ex |
| _ -> raise (Failure ("first parameter in prod should be assign")) |
| in |
| 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 |
| 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 |
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, m_set, l_count, c_count, i_list) = eval (tmp_v_set, m_set, l_count, c_count, i_list) 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, m_set, l_count, c_count, i_list) |

_ -> raise (Failure ("no matrix in prod"))

| Literal(f) -> Float, (v_set, m_set, l_count, c_count, i_list)

| MLiteral(m) -> Matrix, (v_set, m_set, l_count, c_count, i_list)

| Id(s) -> if StringSet.mem s v_set then

Float, (v_set, m_set, l_count, c_count, i_list)

else if StringSet.mem s m_set then

Matrix, (v_set, m_set, l_count, c_count, i_list)

else raise (Failure ("Undeclared identifier " ^ s))

| 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

match v with

Matrix ->

(match op with

Trans -> Matrix, (v_set, m_set, l_count, c_count, i_list)

_ -> raise (Failure ("Un-supported matrix unit operator"))

)

| Float ->

(match op with

Sin -> Float, (v_set, m_set, l_count, c_count, i_list)

| Cos -> Float, (v_set, m_set, l_count, c_count, i_list)

| Log -> Float, (v_set, m_set, l_count, c_count, i_list)

| Tan -> Float, (v_set, m_set, l_count, c_count, i_list)

_ -> raise (Failure ("Un-supported float unit operator")

)

| Assign(vid, e) ->

if StringSet.mem vid func_name then raise (Failure ("Cannot assign a new var with the same name of a formular"))

else

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

match v with

Float -> if StringSet.mem vid m_set then raise (Failure ("Cannot assign a new var with the same name of a matrix"))

else

let v_set = StringSet.add vid v_set in
Float, (v_set, m_set, l_count, c_count, i_list)

| Matrix -> if StringSet.mem vid v_set then raise (Failure (
| "Cannot assign a var with a matrix")) else
let m_set = StringSet.add vid m_set in
Matrix, (v_set, m_set, l_count, c_count, i_list)
)

(* Capture function calls and logical validations in top-
level and handle printing *)
let eval_top (v_set, m_set, l_count, c_count, i_list) (expr :
expr) =
(match expr with
Call(f, para_list) -> let fdecl =
try
FPMap.find (f, List.length para_list) func_decls
with Not_found -> raise (Failure ("Undefined function " ^ f)
)
let g_list = StringMap.find f func_vars
in
List.iter (fun g -> if StringSet.mem g v_set then () else
raise (Failure ("Uninitialized global variable " ^ g)) )
g_list;
if (List.length fdecl.parameter) <> (List.length para_list)
then raise (Failure ("Unmatched parameters"))
else
let (v_set, m_set, l_count, c_count, i_list) = List.
fold_left
(fun (v_set, m_set, l_count, c_count, i_list) actual ->
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
match v with
Float -> (v_set, m_set, l_count, c_count, i_list)
| Matrix -> raise (Failure ("Formula can not take in matrix ")
)
(v_set, m_set, l_count, c_count, i_list) (List.rev para_list
)

let display = if (List.length para_list) > 0 then
"string ("\"\") ^ "^\"" ~ f ~ (" ~ String.escaped(
  expr_list_to_string para_list) ~ ")=" ~ f ~ (\"^+
^ (expr_list_to_string_p "^\", \"^" para_list) ~ "+")=\"^+
" dtos(" ~ f " (" ~ (expr_list_to_string_c para_list) ~ ")")"
else "string(\"\")" ^ "\" +" ^ f ^ "(\"String.escaped(  
   expr_list_to_string para_list) ^ \")\" +" ^ f ^ "("  
   (expr_list_to_string_p "+\", \"+\" para_list) ^ "+\")=\"+  
dtos(" +" ^ f ^ "("  
   (expr_list_to_string_c para_list) ^ "))\")"  
in  
let c_count = c_count + 1  
in  
let instruction = "c_result[" ^ (string_of_int (c_count-1))  
   ^ "]=" ^ display ^ ";"  
in  
let i_list = instruction :: i_list  
in  
(v_set, m_set, l_count, c_count, i_list)  
| 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  
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  
(match (v1, v2) with  
  Equal -> let display_first = "string(\"\")" ^ "\"  
    String.escaped(fst(expr_to_string expr)) ^ "\" in  
let display_left = "dtos(" ^ fst(expr_to_string_c e1) ^ ")\")"  
in  
let display_right = "dtos(" ^ fst(expr_to_string_c e2) ^ ")\")"  
in  
let display_last = "(" ^ fst(expr_to_string_c expr) ^  
   "\" true\"/\" false\"\" ^ ")\") in 
(match op with  
  Equal -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "\" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| Neq -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| Less -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| NotEqual -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| GreaterEqual -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| EqualTo -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)  
| LessEqual -> let display = display_first ^ "\" ^ "\"  
    Rightarrow(" ^ "氘" ^ "\" ^ "氘" ^ "\")  
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)
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
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
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
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)
let i_list = instruction :: i_list in (v_set, m_set, l_count, c_count, i_list)
| _ -> raise (Failure ("Matrix 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 ("Matrix 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)
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("Un-supported matrix unit operator"))
)
)

match op with

Sin -> 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)
| Cos -> 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)
| Log -> 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)
| Tan -> 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("Un-supported float unit operator"))
)
)

match op with

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
(match v with

Float -> if StringSet.mem vid m_set then raise (Failure("Cannot assign a new var with the same name of a matrix"))
else
let v_set = StringSet.add vid v_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)
| Matrix -> if StringSet.mem vid v_set then raise (Failure("Cannot assign a var with a matrix"))
else
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
(* Generate 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 ^ " 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" ^
matrix_class "\n"
"double vdata[NUM_OF_VARIABLES];\n"
"matrix mdata[NUM_OF_MATRIX_VARIABLES];\n"
v_str "\n"
m_str "\n"
dtos "\n"

String.concat "\n" (List.rev f_list) "\n"
main_preamble v_list m_list fdef_latex_l "\n"

String.concat "\n" (List.map (fun ins -> "\t" ^ ins) (List.rev i_list)) "\n"
lfile
in

(* Write string to file *)
let out = open_out cppfile in
fprintf out "%s\n" code;
close_out out;

# File: testall.sh

#!/bin/sh

EZMATH="./../..EZMath"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
keep=0

Usage() {
  echo "Usage: testall.sh [options] [.tex files]"
  echo "-k Keep intermediate files"
  echo "-h Print this help"
  exit 1
}

SignalError() {
  if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
  fi
}
fi

echo "$1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
  echo diff -b "$1" "$2" > "$3" 1>&2
  diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
  }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
  echo $* 1>&2
  eval $*
}

Check() {
  error=0
  basename=`echo $1 | sed 's/.*\///'
  basename=`echo $1 | sed 's/.tex//'`
  echo -n "$basename .tex ..."
  echo "#### Testing $basename" 1>&2
  generatedfiles="output/interpret/${basename}.out"
  referfile="ref/interpret/${basename}.out"
  diff="output/interpret/${basename}.diff"
  Run "$EZMATH" "-i" $1 "1>" $generatedfiles "2>&1"
  Compare $generatedfiles $referfile $diff

  latexfile="output/compile/${basename}.tex"
  referfile="ref/compile/${basename}.tex"
  diff="output/compile/${basename}.diff"
  Run "$EZMATH" "-c" $1 "-l" $latexfile
  Run "g++ -std=c++11 result.cpp" &
  Run ".a.out" &
  Run "rm -f a.out result.cpp" &
  Compare $latexfile $referfile $diff
  if [ $error -eq 0 ]; then
    if [ $keep -eq 0 ]; then
      rm -f $generatedfiles $latexfile
      fi
    fi
}
```bash
75    echo "OK"
76    echo "######## SUCCESS" 1>&2
else
78    echo "######## FAILED" 1>&2
globalerror=$error
81    fi
82
83
84 while getopts kdpsh c; do
85    case $c in
86      k)
87        keep=1
88        ;;
89      h)
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
ocamlex 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:
  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 compile.cmo
    header.cmo ast.cmo
  EZMath.cmx: scanner.cmx parser.cmx interpret.cmx compile.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