YAMML: Yet-Another-Matrix-Manipulation-Language

Name: Bill Chen, Janet Zhang, James Xu, Kent Hall, Doria Chen
Uni: gc2677, jz2896, jrx2000, kjh2166, dc3267

April 21, 2021

Contents

1 Introduction

The rise of machine learning has seen a rise in need of matrix-based computations. Neural Networks used in deep learning, in essence, boil down to a series of matrix operations. Hence, a language that simplifies matrix operations can drastically simplify the workflow for ML engineers and architects, allowing for fast prototyping, ease of use, and high likelihood of correctness. With that goal in mind, we aim to create an

imperative language that has a familiar syntax to existing languages such as Java and C++, while adding built-in support for matrix creation and a series of common matrix operations.

2 Language Tutorial

2.1 Installation

First you have to install OpenCV either through homebrew or your operating system's package manager; make sure the linking paths all work with the ones we have written in the makefile, yammlc-cv.sh and testall.sh files.

2.2 Running the Program

First, compile the compiler with

```
make
```

Then run all the tests by running the shell script to make sure that the compiler is doing its job.

```
./testall.sh
```

Now we are ready to compile our YAMML programs! We provided the a one-step shell script that compiles .yamml files, links with libraries and generates executables. Make a file in the root directory called helloworld.yamml and type the following

```
int main(){
    prints("Hello World!");
}
```

Save the file and head back to the directory. Then run

```
./yammalc-cv ./helloworld.yamml
```

which will generate a .exe executable of the same name. Now run

```
./helloworld.exe
```

Congratulations! You just wrote your first YAMML program.

3 Language Reference Manual

3.1 Data Types

3.1.1 Primitives

Primitives are types with fixed lengths. The 4 primitives are: `int`, `float`, `char`, and `bool`.

3.1.2 Strings

A string is simply a pointer to a contiguous chunk of characters terminated by null terminator.

3.1.3 Matrices

A matrix is a 2d array data container stored contiguously in memory. In essence, it is a struct. In order to make information such as the dimensions of the matrix easily accessible, the struct stores the matrix height and width on the stack. The struct also contains a pointer that points to a contiguous chunk of memory on the heap.
Matrices are initialized by declaration and populated using the square bracket notation. Commas separate elements on the same row, and a semicolon creates a new row in the matrix. Matrices have to be rectangular. Users can index into a matrix as well as slice a matrix using the matrix operators described in Section 3.2.3.2. e.g.

```
matrix M = [2.0,2.0;1.0,2.1]; // matrix declaration
M = [1.1;2.1,3.1]; // Error
M[1,2]; // indexing to value at column 1, row 2
M[0:1,1:1]; // slicing a matrix along i:j,k:l
```

Matrices are mutable and reassignment creates a shallow copy, but slicing them creates separate deep copies.

```
matrix M = empty(5,5); // make a 5 by 5 matrix populated with 0.0
matrix C = M; // reassignment (makes shallow copy of M)
matrix C2 = M[:,:]; // make a deep copy of M
```

Matrices are initialized by declaration and populated using the square bracket notation:

```
int h = 2;
int w = 2;
matrix M = empty(h, w);
M = [1,2;3,4]; // [1,2] and [3,4] are two different rows respectively
M = [1;2,3]; // Error
```

3.2 Lexical Conventions

3.2.1 Identifiers

Valid identifiers are made from ASCII letters and decimal digits. An identifier must begin with a letter, can contain an underscore, cannot be a YAMML keyword.

```
// Valid identifiers
thisisfine
this_is_fine
This_is_FINE
FINE

// invalid identifiers
123id
0
```

3.2.2 Keywords

Keywords are reserved identifiers, They cannot be used as ordinary identifiers for other purposes. YAMML keywords are:

```
if else for while continue break return import
  int char str bool float
  height width
 true false
```

3.2.3 Operators

YAMML uses the following operators

3.2.3.1 Basic Operators

Operator	Meaning	Notes
=	assignment	
+ - * /	arithmetic	Note: operations between ints and floats will have float results
== != <><= >=	comparison	
! && ||	logical	

3.2.3.2 Matrix Operators

Operator	Meaning	Notes
+ -	point-wise addition and subtraction	
*	matrix multiplication	The following mixed binary operations are allowed and are commutative: matrix * float
.* ./	element-wise multiplication and division	with floats
$M[a, b]$	index a matrix by row a and column b	Index starts at 0. M can be any expression that returns a matrix or just a matrix literal such as [1.1, 2.1].
$M[i : j, k : l]$	slice a matrix between rows i and j and columns k and l	the j and l values are inclusive

3.2.3.3 Precedence and Associativity

From lowest to highest precedence, the table below lists the operators and their associativities:

Operator	Meaning	Associativity
=	Assignment	Right
$\|\|$, &&	Or/And	Left
==, !=	Equality/Inequality	Left
$<=, >=, <, >$	Comparison	Left
$+, -$	Addition/Subtraction	Left
$*, /, .*, ./$	Multiplication/Division/Element-wise Matrix Multiplication/Division	Left
$M[i, j]$	Matrix Indexing	Right
$M[i : j, k : l]$	Matrix Slicing	Right
!	Negation/Not	Right

3.2.4 Separators

YAMML uses parentheses to override the default precedence of expression evaluation. Parentheses are also used to separate conditions in loops. Semicolons separate expressions, which are the simplest of statements;

```
int i = 3 + (5 - 6);
for (i; i < 10; i = i + 1){
    print(i);
}
```

3.2.5 Comments

YAMML has both single-line and multi-line comments. The style is identical to that of C's: Any token followed by // are considered part of a single-line comment and are not lexed or parsed.

inline comment style	//this is an inline comment
multiline comment style	/* this is a multi-line comment */

3.2.6 Imports

Functioning similarly to C "includes", `#import` allows users to use globally-exposed functions and variables from other yamml files. For example:

```
#import <file.yamml>
```

3.3 Syntax

3.3.1 Statements

A YAMML program is made up of statements of the following types:

```
If-Else statement
While loop
For loop
Function Definitions
Expression
```

3.3.1.1 If...Else Statements

```
int a = 1;
int b = 3;
if (a <= b){
    print(a);
}
else{
    print(b);
}
```

3.3.1.2 While Statements

```
int i = 0;
while (i < 10){
    i = i + 1;
}
```

3.3.1.3 For Statements

```
int i = 0;
for (i;i < 10; i = i + 1){
    print(i);
}
```

3.3.2 Expressions

Expressions are parts of statements, which are evaluated into expressions and variables using operators. These can be arithmetic expressions which includes an operand, an arithmetic operator, and a function call, which calls a function and returns a value.

3.3.3 Literals

Literals represents strings or one of YAMML's primitive types: float, char, int, boolean and strings

3.3.3.1 Float Literals A float literal is a number written as a whole number with an optional decimal point, a following fraction, and an optional exponent

```
((([0-9]+\.[0-9]*)|([0-9]*\.[0-9]+))((e|E)(\+|-)?[0-9]+)?|[0-9]+((e|E)(\+|-)?[0-9]+))

// Examples
12
1.2
0.14159
12.
1e+3
12.5e-5
```

3.3.3.2 Char Literal A char literal contains only one character surrounded by single quotes assigned to a variable of type char

```
char x;
x = ‘a’;
```

3.3.3.3 String Literals A string literal is a sequence of characters enclosed in double quotation marks

```
(".*[^\\]")
"This is also a string literal"
```

3.3.3.4 Int Literals Int literals are a sequence of integers between 0 and 9.

```
[0-9]+
```

3.3.3.5 Boolean literals Boolean literals use true or false keywords to represent true or false values.

3.3.4 Brackets

YAMML uses curly brackets {} to determine the grouping of statements. This behavior is identical to C. A grouping of statement forms a compound statement, which can be used where one is expected. A compound statement will have the form:

```
{ statement-list }
```

3.4 Functions

3.4.1 Function Calls

Functions are called by name and their argument in parentheses.

```
returned_obj_type function_name (argument_type arg_name)
```

User defined functions from other files need to be imported at the beginning of the file in order to be called.

```
// importing file with user defined function add_one() from above example
# import <math_operations.yamml>;

// calling user defined function with appropriate argument
add_one(1);
```

3.4.2 Variable Assignment from Functions

A function can be the right hand value of an assignment operator. In such a case, the left hand value of the assignment operator is assigned with the return value of the function on the right hand side.

```
# import <math_operations.yamml>;

int x = add_one(1); // x is assigned with the return value, 2, of function add_one()
```

3.4.3 User Defined Functions

Similar to C, YAMML supports user-defined functions. The user must specify the input and return types:

```
// math_operations.yamml
int add_one (int a)
    return a+1;
```

3.4.4 Function Return

Functions return to caller by means of the return statement. YAMML functions can return any of YAMML's data types: `int`, `float`, `char`, `bool`, `str`, and `matrix`.

```
int x = 5;
return x;
```

3.5 Standard Library

3.5.1 Objects

YAMML has two types of built-in objects: matrices and strings. We include the following built-in functions for the matrix object to facilitate matrix algorithms.

3.5.2 Print Functions

Each data type has its own version of print.

Definition	Description	Notes
print(int)	prints an integer	
printf(float)	prints a float	
printb(bool)	prints a boolean as 0 (false) or 1 (true)	
prints(str)	prints a string	
printm(matrix)	prints a matrix	Each row is separated by a new line and each column is separated by a tab, i.e. [0 1 2 3 4 5]

3.5.3 Matrix Functions

Definition	Description
int height(matrix)	returns the height of the matrix, i.e. number of rows
int width(matrix)	returns the width of the matrix, i.e. number of columns
str printm(matrix)	print the matrix
float sum(matrix)	sum of all the elements in the matrix
float mean(matrix)	average of all the elements in the matrix
matrix invert(matrix)	inverse of matrix
matrix filter2D(matrix, matrix, int, matrix)	convolve matrix with kernel (OpenCV wrapper)
matrix trans(matrix)	returns the transposed matrix
matrix empty(int, int)	creates a matrix with passed dimensions, filled with zeros
matrix imread(str)	reads an image's pixel values as a matrix (OpenCV wrapper)
void imwrite(str, matrix)	writes a matrix of pixel values as an image file (OpenCV wrapper)
matrix pow(matrix m, int exp)	Standard library proof of concept written in yamml

3.5.4 Casting

The user can cast variables of one type to another. Because YAMML is strong-typed like C, all casts must be explicit. The syntax for type casting is similar to C:

```
var2 = (type) var1
```

The following casts are allowed:

```
(int) float -> int
(float) int -> float
(char) int -> char
(char) float -> char
```

The casting semantic is the exact same as C.

3.6 Code Examples

The following example applies a sobel operator to detect edges in a certain direction for an image

```
int main()
{
 matrix M = imread("./doge.jpeg");
 matrix sobel_kernel = [1.0,0.0,-1.0;2.0,0.0,-2.0;1.0,0.0,-1.0] ;
 matrix gaussian_kernel = [1.0,2.0,1.0;2.0,4.0,2.0;1.0,2.0,1.0] * (1.0/16.0);
 matrix processed =  empty(height(M), width(M));
 filter2D(M, processed, -1, trans(sobel_kernel));
 imwrite("newhoug.png", processed);

}
```

Figure 1: Before Processing

Figure 2: After Processing

3.7 References

1. http://www.cs.columbia.edu/ sedwards/classes/2018/4115-fall/lrms/Coral.pdf
2. http://www.cs.columbia.edu/ sedwards/classes/2018/4115-fall/reports/MMM.pdf

4 Project Plan

4.1 Process Used

We had weekly hour long meetings every Mondays right after class to identify what we additional features we could implement based on the contents class. During finals week, we met every single day and worked hours unending to finish our language.

4.2 Responsibilities

Note that our responsibilities sort of blurred at the end where everyone became involved in everything. Testers became especially involved in implementations.
Here is the break down

1. **Bill** Project Manager
2. **Kent** System Architect
3. **Janet** Language Guru
4. **James** Testing and Implementing
5. **Doria** Testing and Implementing

4.3 Project Timeline

1. **Proposal Submitted** February 2
2. **LRM and Parser Submitted** February 24
3. **Hello World Working; SAST and Semant framework completed** March 24th
4. **Finished everything; Linked Libraries; Finished Testing Files; Finished Implementing Matrix; Literally everything** April 26th

4.4 Project Log

Note that the project log does not accurately reflect the contribution of each individual. Most of the time we collaborated using Visual Studio Code's LiveShare extension to work on Janet's Machine (it's kind of like a Google Docs for code) so we committed from her machine and under her name disproportionately.

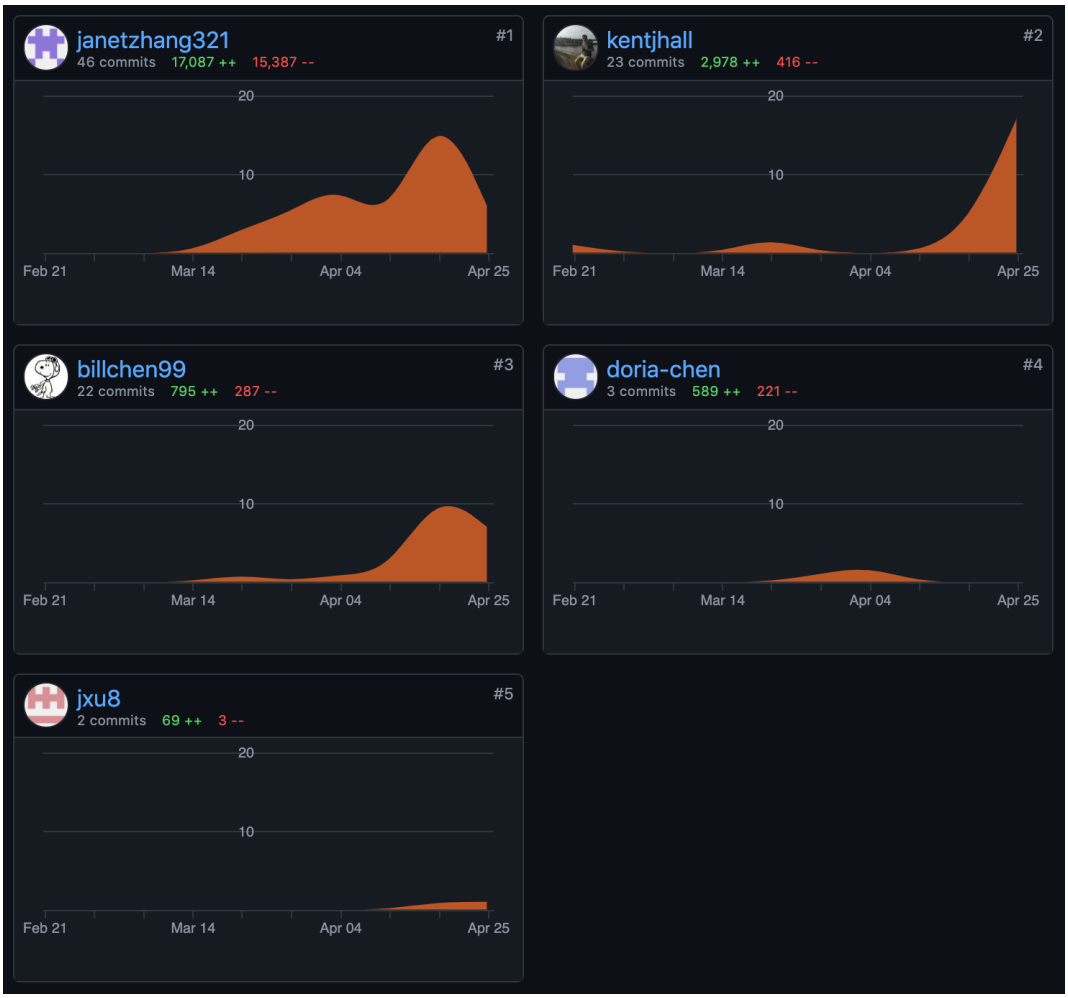
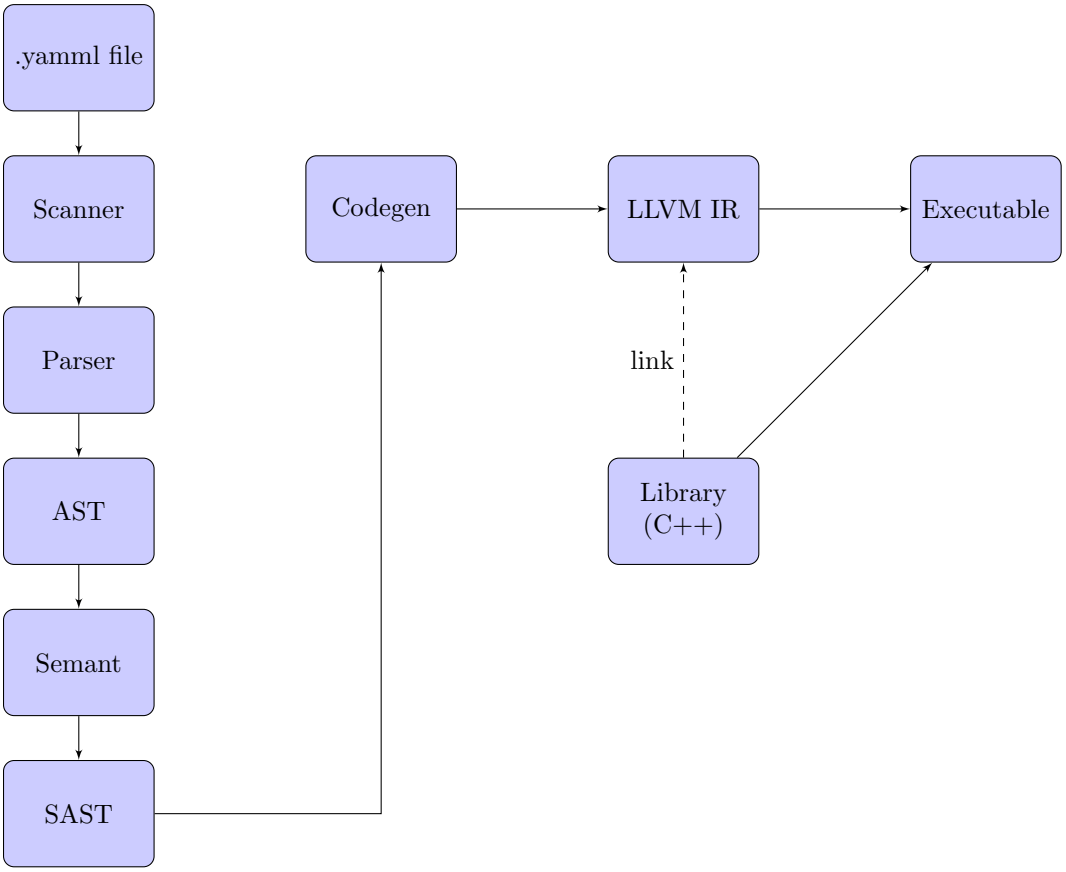


Figure 3: Project Log

5 Architectural Design

5.1 Architecture Diagram



5.2 Program Structure

1. **yamml.ml**
   This is the upper level compiler that executes scanner, parser, semant, and codegen in order to generate the LLVM IR

2. **parser.mly**
   This is the parser definition used by ocamlyacc to generate the parser that parses the scanned tokens to produce an ast

3. **ast.ml**
   This is the abstract syntax tree

4. **sast.ml**
   This is the semantically checked abstract syntax tree.

5. **scanner.mll**
   This is the scanner that turns the program into tokens.

6. **semant.ml**
   This is the semantic checker that takes in the ast and returns a sast to ensure proper typing.

7. **testall.sh**
   This is the driver to perform testing.

8. **codegen.ml**
   The file takes in an SAST and translates the sast into LLVM IR using the LLVM ocaml bindings

5.3 Interface Between Components

As we did not know how to pass a struct into our C++ backend, we had to break up our matrix struct into its respective components (a pointer to the heap, number of rows, and number of columns) and pass it into the backend C++ functions defined externally. To pass into our OpenCV library functions, we also had to write a C++ wrapper for it which we used to link to the OpenCV to our executable.

5.4 Breakdown of Implementation

All members were involved in overall implementation of the project, but some members held larger focus on a certain components. Kent and Janet were crucial contributors to the design and implementation of the parser and scanner as well as local and global variable bindings. Bill and Kent were a key contributors to implementing the matrix access and binary operations. Doria and Janet focused on standard library and function calls. James and Doria also had a focus on creating fail tests to the test suite. Each person was involved in writing their own success tests for the features that they implemented. Implementation was carried out individually or as a group in a shared session where group members were all involved in real-time collaborative implementation.

6 Tests

6.1 Source to Target

6.1.1 Testing Basic Operations

We successfully tested that we could perform arithmetic operations using basic data types. When adding a float and an integer together, the result is a float, so we could use printf to print it (see 3.5.2 Print Functions)

```
int main()
{
    print(1+1); //2
}
int main()
{
    printf(1.0+1.1); //2.1
}
int main()
{
    printf((1+1.1)); //2.1
}
```

6.1.2 Testing Statements

We tested that YAMML can handle conditionals.

```
int main()
{
   if (true) print(42);
}
int main()
{
   if (false) print(42); print(52); //52
}
int main()
```

```
    {
        bool x = false;
        if (x==false) prints("got true, which is what I wanted"); /*"got true, which is what I
            wanted"*/
        print(52); //not printed
    }
\end{verbatim}
\subsubsection{Testing Scope}
We verified that YAMML can create a new scope for local variables using brackets.
\begin{verbatim}
    int main()
    {
            int z;
            int x = 5;
            {
                x = x + 1;
                int y;
                y = 6;
                z = x + y;
            }
            print(z); //12
    }
```

This produces the IR:

```
    ; ModuleID = 'YAMML'
source_filename = "YAMML"

%matrix_t = type { double*, i32, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1

declare i32 @printf(i8*, ...)

declare double @ssum(double*, i32, i32)

declare double @smean(double*, i32, i32)

declare void @strans(double*, double*, i32, i32)

declare void @sfilter2D(%matrix_t*, %matrix_t*, i32, %matrix_t*)

declare void @simread(i8*, %matrix_t*)

declare i1 @simwrite(i8*, %matrix_t*)

declare void @sconsmul(double*, double*, i32, i32, double)

declare void @smatmul(double*, double*, double*, i32, i32, i32, i32)

declare void @selemul(double*, double*, double*, i32, i32)

declare i32 @sprintm(double*, i32, i32)

declare void @smatslice(%matrix_t*, i32, i32, i32, i32, %matrix_t*)

declare void @sinvert(%matrix_t*, %matrix_t*)
```

```
define i32 @main() {
entry:
  %z = alloca i32, align 4
  %x = alloca i32, align 4
  store i32 5, i32* %x, align 4
  %x1 = load i32, i32* %x, align 4
  %tmp = add i32 %x1, 1
  store i32 %tmp, i32* %x, align 4
  %y = alloca i32, align 4
  store i32 6, i32* %y, align 4
  %x2 = load i32, i32* %x, align 4
  %y3 = load i32, i32* %y, align 4
  %tmp4 = add i32 %x2, %y3
  store i32 %tmp4, i32* %z, align 4
  %z5 = load i32, i32* %z, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32
       0, i32 0), i32 %z5)
  ret i32 0
}
```

We tested that YAMML can properly scope global variables as well.

```
float a = 5.0;
int b = 2 + 5 + 7;
str s = "hey world\n";
int main()
{
    float c = 7.8;
    printf(a + c); //12.8
    print(b); //14
}
```

This generates the IR

```
   ; ModuleID = 'YAMML'
source_filename = "YAMML"

%matrix_t = type { double*, i32, i32 }

@.str = global [11 x i8] c"hey world\0A\00"
@s = global i8* getelementptr inbounds ([11 x i8], [11 x i8]* @.str, i64 0, i64 0)
@b = global i32 14
@a = global double 5.000000e+00
@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1

declare i32 @printf(i8*, ...)

declare double @ssum(double*, i32, i32)

declare double @smean(double*, i32, i32)

declare void @strans(double*, double*, i32, i32)

declare void @sfilter2D(%matrix_t*, %matrix_t*, i32, %matrix_t*)

declare void @simread(i8*, %matrix_t*)
```

```
declare i1 @simwrite(i8*, %matrix_t*)

declare void @sconsmul(double*, double*, i32, i32, double)

declare void @smatmul(double*, double*, double*, i32, i32, i32, i32)

declare void @selemul(double*, double*, double*, i32, i32)

declare i32 @sprintm(double*, i32, i32)

declare void @smatslice(%matrix_t*, i32, i32, i32, i32, %matrix_t*)

declare void @sinvert(%matrix_t*, %matrix_t*)

define i32 @main() {
entry:
  %c = alloca double, align 8
  store double 0x401F333333333333, double* %c, align 8
  %a = load double, double* @a, align 8
  %c1 = load double, double* %c, align 8
  %tmp = fadd double %a, %c1
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32
       0, i32 0), double %tmp)
  %b = load i32, i32* @b, align 4
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
      i32 0, i32 0), i32 %b)
  ret i32 0
}
```

6.1.3 Testing Matrix Operations

We tested that users can access elements using a matrix variable (1) and directly from a matrix (2). We also verified that functions can return matrices and that users can perform access operations on the result of the function return (3).

```
matrix myfunc()
{
return [ 0.0, 2.0, 5.1 ];
}

int main() {
    matrix M = [1.0, 1.1];
    printf(M[0,1]);
    printf([1.0, 2.1][0,1]);
printf(myfunc()[0,2]);
}
```

This returns the result 1.1, 2.1 and 5.1 and generates the IR:

```
   ; ModuleID = 'YAMML'
source_filename = "YAMML"

%matrix_t = type { double*, i32, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
```

```
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.6 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.7 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1

declare i32 @printf(i8*, ...)

declare double @ssum(double*, i32, i32)

declare double @smean(double*, i32, i32)

declare void @strans(double*, double*, i32, i32)

declare void @sfilter2D(%matrix_t*, %matrix_t*, i32, %matrix_t*)

declare void @simread(i8*, %matrix_t*)

declare i1 @simwrite(i8*, %matrix_t*)

declare void @sconsmul(double*, double*, i32, i32, double)

declare void @smatmul(double*, double*, double*, i32, i32, i32, i32)

declare void @selemul(double*, double*, double*, i32, i32)

declare i32 @sprintm(double*, i32, i32)

declare void @smatslice(%matrix_t*, i32, i32, i32, i32, %matrix_t*)

declare void @sinvert(%matrix_t*, %matrix_t*)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double, double*
       null, i32 1) to i32), i32 2))
  %system_mat = bitcast i8* %malloccall to double*
  %element_ptr = getelementptr double, double* %system_mat, i32 0
  store double 1.000000e+00, double* %element_ptr, align 8
  %element_ptr1 = getelementptr double, double* %system_mat, i32 1
  store double 1.100000e+00, double* %element_ptr1, align 8
  %m = alloca %matrix_t, align 8
  %m_mat = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 0
  store double* %system_mat, double** %m_mat, align 8
  %m_r = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 1
  store i32 1, i32* %m_r, align 4
  %m_c = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 2
  store i32 2, i32* %m_c, align 4
  %M = load %matrix_t, %matrix_t* %m, align 8
  %M2 = alloca %matrix_t, align 8
  store %matrix_t %M, %matrix_t* %M2, align 8
  %m_mat3 = getelementptr inbounds %matrix_t, %matrix_t* %M2, i32 0, i32 0
  %mat = load double*, double** %m_mat3, align 8
  %m_c4 = getelementptr inbounds %matrix_t, %matrix_t* %M2, i32 0, i32 2
  %c_mat = load i32, i32* %m_c4, align 4
  %tmp = mul i32 0, %c_mat
  %index = add i32 1, %tmp
  %element_ptr_ptr = getelementptr double, double* %mat, i32 %index
  %element_ptr5 = load double, double* %element_ptr_ptr, align 8
```

```
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32
       0, i32 0), double %element_ptr5)
  %malloccall6 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double, double
      * null, i32 1) to i32), i32 2))
  %system_mat7 = bitcast i8* %malloccall6 to double*
  %element_ptr8 = getelementptr double, double* %system_mat7, i32 0
  store double 1.000000e+00, double* %element_ptr8, align 8
  %element_ptr9 = getelementptr double, double* %system_mat7, i32 1
  store double 2.100000e+00, double* %element_ptr9, align 8
  %m10 = alloca %matrix_t, align 8
  %m_mat11 = getelementptr inbounds %matrix_t, %matrix_t* %m10, i32 0, i32 0
  store double* %system_mat7, double** %m_mat11, align 8
  %m_r12 = getelementptr inbounds %matrix_t, %matrix_t* %m10, i32 0, i32 1
  store i32 1, i32* %m_r12, align 4
  %m_c13 = getelementptr inbounds %matrix_t, %matrix_t* %m10, i32 0, i32 2
  store i32 2, i32* %m_c13, align 4
  %m_mat14 = getelementptr inbounds %matrix_t, %matrix_t* %m10, i32 0, i32 0
  %mat15 = load double*, double** %m_mat14, align 8
  %m_c16 = getelementptr inbounds %matrix_t, %matrix_t* %m10, i32 0, i32 2
  %c_mat17 = load i32, i32* %m_c16, align 4
  %tmp18 = mul i32 0, %c_mat17
  %index19 = add i32 1, %tmp18
  %element_ptr_ptr20 = getelementptr double, double* %mat15, i32 %index19
  %element_ptr21 = load double, double* %element_ptr_ptr20, align 8
  %printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
      i32 0, i32 0), double %element_ptr21)
  %.ret = alloca %matrix_t, align 8
  %myfunc_result = call %matrix_t @myfunc()
  store %matrix_t %myfunc_result, %matrix_t* %.ret, align 8
  %m_mat23 = getelementptr inbounds %matrix_t, %matrix_t* %.ret, i32 0, i32 0
  %mat24 = load double*, double** %m_mat23, align 8
  %m_c25 = getelementptr inbounds %matrix_t, %matrix_t* %.ret, i32 0, i32 2
  %c_mat26 = load i32, i32* %m_c25, align 4
  %tmp27 = mul i32 0, %c_mat26
  %index28 = add i32 2, %tmp27
  %element_ptr_ptr29 = getelementptr double, double* %mat24, i32 %index28
  %element_ptr30 = load double, double* %element_ptr_ptr29, align 8
  %printf31 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
      i32 0, i32 0), double %element_ptr30)
  ret i32 0
}

define %matrix_t @myfunc() {
entry:
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double, double*
       null, i32 1) to i32), i32 3))
  %system_mat = bitcast i8* %malloccall to double*
  %element_ptr = getelementptr double, double* %system_mat, i32 0
  store double 0.000000e+00, double* %element_ptr, align 8
  %element_ptr1 = getelementptr double, double* %system_mat, i32 1
  store double 2.000000e+00, double* %element_ptr1, align 8
  %element_ptr2 = getelementptr double, double* %system_mat, i32 2
  store double 5.100000e+00, double* %element_ptr2, align 8
  %m = alloca %matrix_t, align 8
  %m_mat = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 0
  store double* %system_mat, double** %m_mat, align 8
  %m_r = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 1
  store i32 1, i32* %m_r, align 4
  %m_c = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 2
```

```
  store i32 3, i32* %m_c, align 4
  %.matlit = load %matrix_t, %matrix_t* %m, align 8
  ret %matrix_t %.matlit
}

declare noalias i8* @malloc(i32)
\end{verbatim}
Testing Matrix multiplication:
\begin{verbatim}
    int main () {
    matrix M = [1.0,2.0,3.0,4.0];
    matrix B = M * 3.0;
    matrix C = 3.0 * M;
    printf(B[0,0]);
    printf(B[0,1]);
    printf(B[0,2]);
    printf(B[0,3]);
    printf(C[0,0]);
    printf(C[0,1]);
    printf(C[0,2]);
    printf(C[0,3]);

}
```

We also verified the performance and accuracy of various standard library functions for the matrix.

```
    int main() {
        matrix M = [1.0, 1.1, 1.2; 1.1, 1.1, 1.2];
        print(width(M));
        print(height(M));
        matrix M = [1.3, 1.0, 4.0];
        printf(mean(M));
        matrix M = [1.4, 1.1];
        printf(sum(M));
        matrix M = [1.0, 1.1, 1.2];
        printf(M[0,2]);
        print(width(M));
        print(height(M));

        matrix B = trans(M);

        printf(B[2,0]);
        print(width(B));
        print(height(B));
        printf(B[1,0]);
        printf(M[0,1]);

        if (B[1, 0] == M[0, 1]){
            prints("True");
        }
        else{
            prints("False");
        }
}
```

The IR

```
    ; ModuleID = 'YAMML'
```

```
source_filename = "YAMML"

%matrix_t = type { double*, i32, i32 }

@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%c\0A\00", align 1
@.strlit = private unnamed_addr constant [5 x i8] c"True\00", align 1
@.strlit.4 = private unnamed_addr constant [6 x i8] c"False\00", align 1

declare i32 @printf(i8*, ...)

declare double @ssum(double*, i32, i32)

declare double @smean(double*, i32, i32)

declare void @strans(double*, double*, i32, i32)

declare void @sfilter2D(%matrix_t*, %matrix_t*, i32, %matrix_t*)

declare void @simread(i8*, %matrix_t*)

declare i1 @simwrite(i8*, %matrix_t*)

declare void @sconsmul(double*, double*, i32, i32, double)

declare void @smatmul(double*, double*, double*, i32, i32, i32, i32)

declare void @selemul(double*, double*, double*, i32, i32)

declare i32 @sprintm(double*, i32, i32)

declare void @smatslice(%matrix_t*, i32, i32, i32, i32, %matrix_t*)

declare void @sinvert(%matrix_t*, %matrix_t*)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double, double*
       null, i32 1) to i32), i32 6))
  %system_mat = bitcast i8* %malloccall to double*
  %element_ptr = getelementptr double, double* %system_mat, i32 0
  store double 1.000000e+00, double* %element_ptr, align 8
  %element_ptr1 = getelementptr double, double* %system_mat, i32 1
  store double 1.100000e+00, double* %element_ptr1, align 8
  %element_ptr2 = getelementptr double, double* %system_mat, i32 2
  store double 1.200000e+00, double* %element_ptr2, align 8
  %element_ptr3 = getelementptr double, double* %system_mat, i32 3
  store double 1.100000e+00, double* %element_ptr3, align 8
  %element_ptr4 = getelementptr double, double* %system_mat, i32 4
  store double 1.100000e+00, double* %element_ptr4, align 8
  %element_ptr5 = getelementptr double, double* %system_mat, i32 5
  store double 1.200000e+00, double* %element_ptr5, align 8
  %m = alloca %matrix_t, align 8
  %m_mat = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 0
  store double* %system_mat, double** %m_mat, align 8
  %m_r = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 1
  store i32 2, i32* %m_r, align 4
```

```
%m_c = getelementptr inbounds %matrix_t, %matrix_t* %m, i32 0, i32 2
store i32 3, i32* %m_c, align 4
%M = load %matrix_t, %matrix_t* %m, align 8
%M6 = alloca %matrix_t, align 8
store %matrix_t %M, %matrix_t* %M6, align 8
%m_c7 = getelementptr inbounds %matrix_t, %matrix_t* %M6, i32 0, i32 2
%c_mat = load i32, i32* %m_c7, align 4
%printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1, i32
     0, i32 0), i32 %c_mat)
%m_r8 = getelementptr inbounds %matrix_t, %matrix_t* %M6, i32 0, i32 1
%r_mat = load i32, i32* %m_r8, align 4
%printf9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
    i32 0, i32 0), i32 %r_mat)
%malloccall10 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double,
    double* null, i32 1) to i32), i32 3))
%system_mat11 = bitcast i8* %malloccall10 to double*
%element_ptr12 = getelementptr double, double* %system_mat11, i32 0
store double 1.300000e+00, double* %element_ptr12, align 8
%element_ptr13 = getelementptr double, double* %system_mat11, i32 1
store double 1.000000e+00, double* %element_ptr13, align 8
%element_ptr14 = getelementptr double, double* %system_mat11, i32 2
store double 4.000000e+00, double* %element_ptr14, align 8
%m15 = alloca %matrix_t, align 8
%m_mat16 = getelementptr inbounds %matrix_t, %matrix_t* %m15, i32 0, i32 0
store double* %system_mat11, double** %m_mat16, align 8
%m_r17 = getelementptr inbounds %matrix_t, %matrix_t* %m15, i32 0, i32 1
store i32 1, i32* %m_r17, align 4
%m_c18 = getelementptr inbounds %matrix_t, %matrix_t* %m15, i32 0, i32 2
store i32 3, i32* %m_c18, align 4
%M19 = load %matrix_t, %matrix_t* %m15, align 8
%M20 = alloca %matrix_t, align 8
store %matrix_t %M19, %matrix_t* %M20, align 8
%m_mat21 = getelementptr inbounds %matrix_t, %matrix_t* %M20, i32 0, i32 0
%mat_mat = load double*, double** %m_mat21, align 8
%m_r22 = getelementptr inbounds %matrix_t, %matrix_t* %M20, i32 0, i32 1
%r_mat23 = load i32, i32* %m_r22, align 4
%m_c24 = getelementptr inbounds %matrix_t, %matrix_t* %M20, i32 0, i32 2
%c_mat25 = load i32, i32* %m_c24, align 4
%mean = call double @smean(double* %mat_mat, i32 %r_mat23, i32 %c_mat25)
%printf26 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %mean)
%malloccall27 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double,
    double* null, i32 1) to i32), i32 2))
%system_mat28 = bitcast i8* %malloccall27 to double*
%element_ptr29 = getelementptr double, double* %system_mat28, i32 0
store double 1.400000e+00, double* %element_ptr29, align 8
%element_ptr30 = getelementptr double, double* %system_mat28, i32 1
store double 1.100000e+00, double* %element_ptr30, align 8
%m31 = alloca %matrix_t, align 8
%m_mat32 = getelementptr inbounds %matrix_t, %matrix_t* %m31, i32 0, i32 0
store double* %system_mat28, double** %m_mat32, align 8
%m_r33 = getelementptr inbounds %matrix_t, %matrix_t* %m31, i32 0, i32 1
store i32 1, i32* %m_r33, align 4
%m_c34 = getelementptr inbounds %matrix_t, %matrix_t* %m31, i32 0, i32 2
store i32 2, i32* %m_c34, align 4
%M35 = load %matrix_t, %matrix_t* %m31, align 8
%M36 = alloca %matrix_t, align 8
store %matrix_t %M35, %matrix_t* %M36, align 8
%m_mat37 = getelementptr inbounds %matrix_t, %matrix_t* %M36, i32 0, i32 0
```

```
%mat_mat38 = load double*, double** %m_mat37, align 8
%m_r39 = getelementptr inbounds %matrix_t, %matrix_t* %M36, i32 0, i32 1
%r_mat40 = load i32, i32* %m_r39, align 4
%m_c41 = getelementptr inbounds %matrix_t, %matrix_t* %M36, i32 0, i32 2
%c_mat42 = load i32, i32* %m_c41, align 4
%sum = call double @ssum(double* %mat_mat38, i32 %r_mat40, i32 %c_mat42)
%printf43 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %sum)
%malloccall44 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr (double,
    double* null, i32 1) to i32), i32 3))
%system_mat45 = bitcast i8* %malloccall44 to double*
%element_ptr46 = getelementptr double, double* %system_mat45, i32 0
store double 1.000000e+00, double* %element_ptr46, align 8
%element_ptr47 = getelementptr double, double* %system_mat45, i32 1
store double 1.100000e+00, double* %element_ptr47, align 8
%element_ptr48 = getelementptr double, double* %system_mat45, i32 2
store double 1.200000e+00, double* %element_ptr48, align 8
%m49 = alloca %matrix_t, align 8
%m_mat50 = getelementptr inbounds %matrix_t, %matrix_t* %m49, i32 0, i32 0
store double* %system_mat45, double** %m_mat50, align 8
%m_r51 = getelementptr inbounds %matrix_t, %matrix_t* %m49, i32 0, i32 1
store i32 1, i32* %m_r51, align 4
%m_c52 = getelementptr inbounds %matrix_t, %matrix_t* %m49, i32 0, i32 2
store i32 3, i32* %m_c52, align 4
%M53 = load %matrix_t, %matrix_t* %m49, align 8
%M54 = alloca %matrix_t, align 8
store %matrix_t %M53, %matrix_t* %M54, align 8
%m_mat55 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 0
%mat = load double*, double** %m_mat55, align 8
%m_c56 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 2
%c_mat57 = load i32, i32* %m_c56, align 4
%tmp = mul i32 0, %c_mat57
%index = add i32 2, %tmp
%element_ptr_ptr = getelementptr double, double* %mat, i32 %index
%element_ptr58 = load double, double* %element_ptr_ptr, align 8
%printf59 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %element_ptr58)
%m_c60 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 2
%c_mat61 = load i32, i32* %m_c60, align 4
%printf62 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
    i32 0, i32 0), i32 %c_mat61)
%m_r63 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 1
%r_mat64 = load i32, i32* %m_r63, align 4
%printf65 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
    i32 0, i32 0), i32 %r_mat64)
%m_mat66 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 0
%mat67 = load double*, double** %m_mat66, align 8
%m_r68 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 1
%r_mat69 = load i32, i32* %m_r68, align 4
%m_c70 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 2
%c_mat71 = load i32, i32* %m_c70, align 4
%tmp72 = mul i32 %c_mat71, %r_mat69
%mallocsize = mul i32 %tmp72, ptrtoint (double* getelementptr (double, double* null, i32 1) to
    i32)
%malloccall73 = tail call i8* @malloc(i32 %mallocsize)
%system_mat74 = bitcast i8* %malloccall73 to double*
%m75 = alloca %matrix_t, align 8
%m_mat76 = getelementptr inbounds %matrix_t, %matrix_t* %m75, i32 0, i32 0
store double* %system_mat74, double** %m_mat76, align 8
```

```
%m_r77 = getelementptr inbounds %matrix_t, %matrix_t* %m75, i32 0, i32 1
store i32 %c_mat71, i32* %m_r77, align 4
%m_c78 = getelementptr inbounds %matrix_t, %matrix_t* %m75, i32 0, i32 2
store i32 %r_mat69, i32* %m_c78, align 4
%m_mat79 = getelementptr inbounds %matrix_t, %matrix_t* %m75, i32 0, i32 0
%mat80 = load double*, double** %m_mat79, align 8
call void @strans(double* %mat67, double* %mat80, i32 %r_mat69, i32 %c_mat71)
%B = load %matrix_t, %matrix_t* %m75, align 8
%B81 = alloca %matrix_t, align 8
store %matrix_t %B, %matrix_t* %B81, align 8
%m_mat82 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 0
%mat83 = load double*, double** %m_mat82, align 8
%m_c84 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 2
%c_mat85 = load i32, i32* %m_c84, align 4
%tmp86 = mul i32 2, %c_mat85
%index87 = add i32 0, %tmp86
%element_ptr_ptr88 = getelementptr double, double* %mat83, i32 %index87
%element_ptr89 = load double, double* %element_ptr_ptr88, align 8
%printf90 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %element_ptr89)
%m_c91 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 2
%c_mat92 = load i32, i32* %m_c91, align 4
%printf93 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
    i32 0, i32 0), i32 %c_mat92)
%m_r94 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 1
%r_mat95 = load i32, i32* %m_r94, align 4
%printf96 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.1,
    i32 0, i32 0), i32 %r_mat95)
%m_mat97 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 0
%mat98 = load double*, double** %m_mat97, align 8
%m_c99 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 2
%c_mat100 = load i32, i32* %m_c99, align 4
%tmp101 = mul i32 1, %c_mat100
%index102 = add i32 0, %tmp101
%element_ptr_ptr103 = getelementptr double, double* %mat98, i32 %index102
%element_ptr104 = load double, double* %element_ptr_ptr103, align 8
%printf105 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %element_ptr104)
%m_mat106 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 0
%mat107 = load double*, double** %m_mat106, align 8
%m_c108 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 2
%c_mat109 = load i32, i32* %m_c108, align 4
%tmp110 = mul i32 0, %c_mat109
%index111 = add i32 1, %tmp110
%element_ptr_ptr112 = getelementptr double, double* %mat107, i32 %index111
%element_ptr113 = load double, double* %element_ptr_ptr112, align 8
%printf114 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2,
    i32 0, i32 0), double %element_ptr113)
%m_mat115 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 0
%mat116 = load double*, double** %m_mat115, align 8
%m_c117 = getelementptr inbounds %matrix_t, %matrix_t* %B81, i32 0, i32 2
%c_mat118 = load i32, i32* %m_c117, align 4
%tmp119 = mul i32 1, %c_mat118
%index120 = add i32 0, %tmp119
%element_ptr_ptr121 = getelementptr double, double* %mat116, i32 %index120
%element_ptr122 = load double, double* %element_ptr_ptr121, align 8
%m_mat123 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 0
%mat124 = load double*, double** %m_mat123, align 8
%m_c125 = getelementptr inbounds %matrix_t, %matrix_t* %M54, i32 0, i32 2
```

```
  %c_mat126 = load i32, i32* %m_c125, align 4
  %tmp127 = mul i32 0, %c_mat126
  %index128 = add i32 1, %tmp127
  %element_ptr_ptr129 = getelementptr double, double* %mat124, i32 %index128
  %element_ptr130 = load double, double* %element_ptr_ptr129, align 8
  %tmp131 = fcmp oeq double %element_ptr122, %element_ptr130
  br i1 %tmp131, label %then, label %else

merge:                                            ; preds = %else, %then
  ret i32 0

then:                                             ; preds = %entry
  %printf132 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt,
      i32 0, i32 0), i8* getelementptr inbounds ([5 x i8], [5 x i8]* @.strlit, i32 0, i32 0))
  br label %merge

else:                                             ; preds = %entry
  %printf133 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt,
      i32 0, i32 0), i8* getelementptr inbounds ([6 x i8], [6 x i8]* @.strlit.4, i32 0, i32 0))
  br label %merge
}

declare noalias i8* @malloc(i32)
\end{verbatim}
\subsubsection{Testing Functions}

The function below takes two integer arguments and prints the first argument if it is larger than
    the second. If not, it will enter a while loop, printing itself and incrementing until it is
    larger than the original second argument. This test function verifies that loops are working as
     well.
\begin{verbatim}
int greater_than_x(int l, int x) {
    if (l > x){
        return l;
    }
    else {
        while (l <= x){
            print (l);
            l = l + 1;
        }
    return l;
    }
}

int main(){
    print(greater_than_x(3, 5)); // 3 4 5 6
}
\end{verbatim}
Can pass in a matrix as an argument and return a matrix
\begin{verbatim}
float matrix_func_arg(matrix m){
    return m[0,0];
}

int main(){
    matrix M = [1.1, 1.2];
    printf(matrix_func_arg(M));
}
```

6.2 Test Import and Linear Regression Demo

```
// library function
matrix regress(matrix X, matrix Y){
    matrix cov = trans(X) * X;
    matrix inverted_conv = cov[:,:];
    invert(cov, inverted_conv);
    return (inverted_conv * trans(X)) * Y;
}


// main file
    #import <linear-regression.yamml>

int main(){
    matrix A = [1.0,1.0,2.0;1.0,3.0,4.0;1.0,5.0,6.0;1.0,10.0,10.4];
    matrix B = [10.0;14.0;18.3;20.5];
    printm(regress(A,B));
}
```

6.3 Test automation

Test automation was mostly just copied over from the MicroC suite. We made slight modifications for 1) linking in OpenCV and 2) testing exit status of a program.

7 Lessons Learned

7.1 Bill Chen

Important things need to be said multiple times: start early, start early, start early! A lot of things will go wrong. I learned how to build a compiler but it was more important that I stood the importance of managing time and team members effectively and assign tasks that can be done in a parallel fashion. I also discovered the difficulties of collaborating remotely which was why we shifted to an in person collaboration mode toward the end. I also needed to have a good overall understanding of the stack in order to do so and spent a lot of time going over the entire stack. Language-wise Ocaml was a pain to pick up in the beginning but it soon became clear the benefits of being able to know at compile time which patterns were unmatched and unused. Being able to think functionally has brought me certain degree of joy in additional understanding. Having a solid test suite will help a lot in making sure newly implemented features does not break the old ones. I suppose the final lesson I learned was to not judge a language by the difficulty

7.2 James Xu

The material was difficult, but I learned a great deal from my teammates. I'm grateful that I had knowledgeable teammates who could help answer my questions. I've learned that ensuring proper and open communication throughout the course of a project is crucial to effective collaboration. It's important to keep each other updated to improve efficiency. As a tester, I found it was important to ask my teammates questions, even at times incessantly. By making sure that I was on the same page, I was able to work more effectively with my team.

7.3 Kent Hall

I think the main thing I took away from this project was to think "functionally"—that is, to become comfortable with functional programming in OCaml. There was a steep learning curve to reasoning about problems recursively in this way, no longer being able to rely on global state, etc. But towards the end, I found

myself increasingly having these "aha" moments where things began clicking into place, and the benefits of OCaml's restrictions became clear. Once the infrastructure was in place and all the critical compiler errors had been resolved, things tended to just work, and implementing new features became a breeze. The best case of this was when I needed to implement"imports" (importing other YAMML files)—I simply turned the existing `ast` variable into a recursive function which takes a filename to parse, and had it `fold_left` to go over all parsed imports, each one recursively calling `ast` and accumulating on the lists of global/function declarations. Accomplishing this task was much simpler and more intuitive than I'd expected, certainly more than if I'd been doing it imperatively in C. I can say confidently that this ended up being the case for the overall project as well, as OCaml's constant pestering about unmatched cases ensured that I would remember to handle various corner cases and unimplemented features.

7.4 Doria Chen

Starting the project can be more intimidating than doing the project, so start the project and start it earlier if possible. I also learned that working-in person with your group is so valuable and can make the process much more efficient. I learned a lot from working on my own to implement things but more from working with my teammates to do so.

7.5 Janet Zhang

Don't take working in-person for granted. It was pretty difficult to coordinate working offline efficiently on a project that requires heavy collaboration. After we began to meet up and pair program, we began fixing problems quickly and got everybody caught up on all the features. Group projects can be pretty fun when you get to know your teammates offline, so I'm grateful to have that experience with everybody on my team.

8 Appendix

8.1 References

The project referred back to MMM and Coral from previous semester's project as a baseline and inspiration for the language and the report.

8.2 Code

Everyone worked on everything so the code attached here is considered signed by everyone:

8.2.1 scanner.mll

```
(* Ocamllex scanner for YAMML *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }     (* Whitespace *)
| "/*"      { block_comment lexbuf }        (* Block Comments *)
| "//"      { sl_comment lexbuf }           (* Single line Comments *)
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LBRACK }
| ']'       { RBRACK }
| ';'       { SEMI }
| ':'       { COLON }
| ','       { COMMA }
```

```
| ".*"      { ELETIMES }
| "./"      { ELEDIVIDE }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "&&"      { AND }
| "||"      { OR }
| "!"       { NOT }
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "return"  { RETURN }
| "continue" { CONTINUE }
| "break"   { BREAK }
| "int"     { INT }
| "bool"    { BOOL }
| "void"    { VOID }
| "char"    { CHAR }
| "str"     { STRING }
| "float"   { FLOAT }
| "matrix"  { MATRIX }
| "true"    { TRUE }
| "false"   { FALSE }
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ''' ([^ ''']* as ch) ''' { CHARLIT(String.get (Scanf.unescaped ch) 0) }
| '"' ([^ '"']* as str) '"' { STRINGLIT(Scanf.unescaped str) }
| ((['0'-'9']+'.'['0'-'9']*)|(['0'-'9']*'.'['0'-'9']+))(('e'|'E')('+'|'-')?['0'-'9']+)
    ?|['0'-'9']+(('e'|'E')('+'|'-')?['0'-'9']+)) as lxm { FLOATLIT(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| [' ' '\t' '\r' '\n']*'#'[' ' '\t' '\r' '\n']*"import"[' ' '\t' '\r' '\n']*'<' ([^ '<' '>']* as im
    ) '>'{ IMPORT(im) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and block_comment = parse
  "*/" { token lexbuf }
| _    { block_comment lexbuf }

and sl_comment = parse
  ['\n' '\r'] { token lexbuf }
| _    { sl_comment lexbuf }
```

8.2.2 parser.mly

```
/* Ocamlyacc parser for YAMML */

%{
open Ast
%}
```

```
%token SEMI COLON LPAREN RPAREN LBRACE RBRACE COMMA
%token LBRACK RBRACK
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token ELETIMES ELEDIVIDE
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE CONTINUE BREAK
%token INT BOOL VOID CHAR STRING FLOAT MATRIX
%token HEIGHT WIDTH
%token <int> LITERAL
%token <char> CHARLIT
%token <string> STRINGLIT
%token <float> FLOATLIT
%token <string> ID
%token <string> IMPORT
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left ELETIMES ELEDIVIDE
%right NOT NEG
%nonassoc LBRACK RBRACK
%nonassoc LPAREN RPAREN

%start program
%type <Ast.program> program

%%

program:
  imports decls EOF { match $2 with (v,f) -> ($1,v,f) }

imports:
   /* nothing */ { [] }
 | imports import { $2 :: $1 }

import:
  IMPORT { Import($1) }

decls:
   /* nothing */ { [], [] }
 | decls vdecl { ($2 :: fst $1), snd $1 }
 | decls fdecl { fst $1, ($2 :: snd $1) }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { typ = $1;
         fname = $2;
         formals = $4;
         body = List.rev $7 } }

formals_opt:
```

```
    /* nothing */ { [] }
  | formal_list  { List.rev $1 }

formal_list:
    typ ID                { [($1,$2)] }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | CHAR { Char }
  | STRING { String }
  | FLOAT { Float }
  | MATRIX { Matrix }

stmt_list:
    /* nothing */ { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr_opt SEMI { Expr $1 }
  | RETURN expr_opt SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | CONTINUE SEMI { Continue }
  | BREAK SEMI { Break }
  | vdecl { VarDecl($1) }

vdecl:
   typ ID SEMI { Decl($1, $2) }
 | typ ID ASSIGN expr SEMI { DeclInit(($1, $2), $4) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr        { $1 }

fst_ind_opt:
    /* nothing */ { Beg }
  | expr        { match $1 with a -> Ind(a) }

snd_ind_opt:
    /* nothing */ { End }
  | expr        { match $1 with a -> Ind(a) }

expr:
    LITERAL        { Literal($1) }
  | TRUE           { BoolLit(true) }
  | FALSE          { BoolLit(false) }
  | CHARLIT        { CharLit($1) }
  | STRINGLIT      { StringLit($1) }
  | FLOATLIT       { FloatLit(string_of_float $1) }
  | ID             { Id($1) }
  | mat_literal    { MatrixLit(fst $1, snd $1) }
  | expr PLUS  expr { Binop($1, Add, $3) }
```

```
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr TIMES expr { Binop($1, Mult, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr ELETIMES expr { Binop($1, Elemult, $3) }
  | expr ELEDIVIDE expr { Binop($1, Elediv, $3) }
  | expr EQ    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 AND   expr { Binop($1, And, $3) }
  | expr OR    expr { Binop($1, Or,  $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr       { Unop(Not, $2) }
  | expr ASSIGN expr { Assign($1, $3) }
  | expr LBRACK expr COMMA expr RBRACK {
    match $3, $5 with
    | r, c -> MatAccess($1,r,c)}
  | expr LBRACK fst_ind_opt COLON snd_ind_opt COMMA fst_ind_opt COLON snd_ind_opt RBRACK {
    match $3, $5, $7, $9 with
    | a,b,c,d -> MatSlice($1,Range(a, b),Range(c, d))}
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | LPAREN typ RPAREN expr { CastExpr($2, $4) }
  | expr LBRACK expr RBRACK { StringAccess($1, $3) }

mat_literal:
    LBRACK RBRACK { [| |], (0, 0) }
  | LBRACK mat_assembly RBRACK { $2 }

mat_assembly:
    ele { [| $1 |], (1, 1) }
  | mat_assembly COMMA ele {
      let a,(r,c) = (fst $1), (snd $1) in
      Array.append a [| $3 |], (r, c+1) }
  | mat_assembly SEMI ele {
      let a,(r,_) = (fst $1), (snd $1) in
      Array.append a [| $3 |], (r+1, 1) }

ele:
    FLOATLIT { $1 }
  | MINUS FLOATLIT %prec NEG { -. $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list { List.rev $1 }

actuals_list:
    expr                 { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

8.2.3 ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Elemult | Elediv
```

```
type uop = Neg | Not

type typ = Int | Bool | Void | Char | String | Float | Matrix | Rangety

type import = Import of string

type expr =
    Literal of int
  | BoolLit of bool
  | CharLit of char
  | StringLit of string
  | FloatLit of string
  | MatrixLit of float array * (int * int)
  | MatAccess of expr * expr * expr
  | MatSlice of expr * expr * expr
  | StringAccess of expr * expr
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | CastExpr of typ * expr
  | Noexpr
  | Range of index * index
  and index = Beg | End | Ind of expr

type bind_decl = typ * string (* (typ, string) *)

type bind_init = bind_decl * expr (* ((typ, string), expr) *)

type bind =
   Decl of bind_decl
 | DeclInit of bind_init

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Continue
  | Break
  | VarDecl of bind

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind_decl list;
    body : stmt list;
  }

type program = import list * bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
```

```
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Elemult -> ".*"
  | Elediv -> "./"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Char -> "char"
  | String -> "str"
  | Float -> "float"
  | Matrix -> "matrix"
  | Rangety -> "range"

let rec string_of_matrix scalar cols _ =
  "[ " ^ (let rec string_of_elems = function
      [] -> ""
    | h::t -> string_of_expr h ^
              (if (List.length t) == 0 then ""
              else if (List.length t) mod cols == 0 then "; "
              else ", ") ^
              string_of_elems t
  in string_of_elems scalar) ^ " ]"

and string_of_expr = function
    Literal(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | CharLit(c) -> "'" ^ String.escaped (String.make 1 c) ^ "'"
  | StringLit(s) -> "\"" ^ String.escaped s ^ "\""
  | FloatLit(f) -> f
  | MatSlice(m, b, c) -> string_of_expr m ^ "[" ^ string_of_expr b ^ "," ^ string_of_expr c ^ "]"
  | MatrixLit(m,(a,b)) ->
     let arr = (List.map (fun x -> FloatLit(Printf.sprintf "%g" x))) (Array.to_list m) in
         string_of_matrix arr a b
  | MatAccess(m, b, c) -> "Accessing " ^ string_of_expr m ^" element [" ^string_of_expr b^ "," ^
      string_of_expr c ^ "]"
  | Range(s, e) ->
     let a = (match s with
         Beg -> "Beg"
       | End -> "End"
       | Ind(e) -> string_of_expr e)
     and b = (match e with
         Beg -> "Beg"
       | End -> "End"
```

```
      | Ind(e) -> string_of_expr e)
    in a ^ " : " ^ b
  | StringAccess(s, i) -> string_of_expr s ^ "[" ^ string_of_expr i ^ "]"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
    string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
    f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | CastExpr(t, e) -> "(" ^ string_of_typ t ^ ") " ^ string_of_expr e
  | Noexpr -> ""

let string_of_vdecl = function
    Decl(decl) -> string_of_typ (fst decl) ^ " " ^ (snd decl) ^ ";\n"
  | DeclInit(decl, exp) -> string_of_typ (fst decl) ^ " " ^ (snd decl) ^ " = " ^ string_of_expr exp
      ^ ";\n"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
    string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
    "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr e2 ^ " ; " ^
    string_of_expr e3 ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Continue -> "continue;\n"
  | Break -> "break;\n"
  | VarDecl(v) -> string_of_vdecl v



  let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

8.2.4 semant.ml

```
(* Semantic checking for the Yamml compiler *)

open Ast
open Sast

module StringHash = Hashtbl.Make(
  struct type t = string (* type of keys *)
  let equal x y = x = y (* use structural comparison *)
  let hash = Hashtbl.hash (* generic hash function *)
```

```
end);;


module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.
   Check each global variable, then check each function *)

let check (globals, functions) =

  let report_duplicate exceptf list =
    let rec helper = function
            n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Verify a list of bindings has no void types or duplicate names *)

  let check_binds (kind : string) (bd : bind_decl list) =
    List.iter (check_not_void (fun n -> "illegal void "^kind^ " " ^n)) bd;
    report_duplicate (fun n -> "duplicate "^ kind ^" " ^ n) (List.map snd bd);

  in

  let rec global_symbols = function
    [] -> []
    | hd::tl -> match hd with
        DeclInit(d) -> fst d::global_symbols tl
      | Decl(d) -> d::global_symbols tl
  in

  (**** Check global variables ****)

  ignore(check_binds "global" (global_symbols globals));

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no bodies *)
  let built_in_decls =
    let add_bind map (name, ty, rty) = StringMap.add name {
      typ = rty;
      fname = name;
      formals = List.fold_left (fun l t -> l @ [(t, "x")]) [] ty;
      body = [] } map
    in List.fold_left add_bind StringMap.empty [
      ("print", [Int], Void);
      ("printf", [Float], Void);
      ("printb", [Bool], Void);
      ("printc", [Char], Void);
      ("printm", [Matrix], Int);
```

```
    ("printbig", [Int], Void);
    ("prints", [String], Void);
    ("height", [Matrix], Int);
    ("width", [Matrix], Int);
    ("sum", [Matrix], Float);
    ("mean", [Matrix], Float);
    ("trans", [Matrix], Matrix);
    ("empty", [Int; Int], Matrix);
    (* OpenCV functions *)
    ("filter2D", [Matrix; Matrix; Int; Matrix], Void);
    ("imread", [String], Matrix);
    ("imwrite", [String; Matrix], Void);
    ("invert", [Matrix; Matrix], Void);
     ]
in

(* Add function name to symbol table *)
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = "duplicate function " ^ fd.fname
  and make_err er = raise (Failure er)
  and n = fd.fname (* Name of the function *)
  in match fd with (* No duplicate functions or redefinitions of built-ins *)
       _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
     | _ -> StringMap.add n fd map
in

(* Collect all function names into one symbol table *)
let function_decls = List.fold_left add_func built_in_decls functions
in

(* Return a function from our symbol table *)
let find_func s =
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

let _ = find_func "main" in (* Ensure "main" is defined *)


(* Return a variable from our local symbol table *)
let type_of_identifier s map =
  try StringMap.find s map
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
in

(* Raise an exception if the given rvalue type cannot be assigned to
   the given lvalue type *)
let check_assign lvaluet rvaluet err =
   if lvaluet = rvaluet then lvaluet else raise (Failure err)
in

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr e map = match e with
     Literal l -> (Int, SLiteral l)
   | FloatLit l -> (Float, SFloatLit l)
   | StringLit l -> (String, SStringLit l)
   | CharLit l -> (Char, SCharLit l)
```

```
| BoolLit l -> (Bool, SBoolLit l)
| MatrixLit(l,(r,c)) ->
   if Array.length l = r * c
   then (Matrix, SMatrixLit(l, (r,c)))
   else raise ( Failure ("illegal Matrix Dimension"))
| Noexpr    -> (Void, SNoexpr)
| Id s      -> (type_of_identifier s map, SId s)
| Assign(var, e) as ex ->
   let lt = (match var with
       Id s -> type_of_identifier s map
     | MatAccess(_,_,_) -> Float
     | StringAccess(_,_) -> Char
     | _ -> raise (Failure "illegal assignment LHS"))
   and (rt, e') = expr e map in
   let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
     string_of_typ rt ^ " in " ^ string_of_expr ex
   in (check_assign lt rt err, SAssign(expr var map, (rt, e')))
| Unop(op, e) as ex ->
   let (t, e') = expr e map in
   let ty = match op with
     Neg when t = Int || t = Float -> t
   | Not when t = Bool -> Bool
   | _ -> raise (Failure ("illegal unary operator " ^
                          string_of_uop op ^ string_of_typ t ^
                          " in " ^ string_of_expr ex))
   in (ty, SUnop(op, (t, e')))
| Binop(e1, op, e2) as e ->
   let (t1, _) = expr e1 map
   and (t2, _) = expr e2 map in
   (* All binary operators require operands of the same type *)
   let same = t1 = t2 in
   (* Determine expression type based on operator and operand types *)
   let ty = match op with
     Add | Sub | Mult | Div when same && t1 = Int -> Int
   | Add | Sub | Mult | Div when same && t1 = Float -> Float
   | Elemult | Elediv | Mult when same && t1 = Matrix -> Matrix
   | Add | Sub | Mult | Div when t1 = Float && t2 = Int -> Float
   | Add | Sub | Mult | Div when t1 = Int && t2 = Float -> Float
   | Equal | Neq          when same             -> Bool
   | Less | Leq | Greater | Geq
            when same && (t1 = Int || t1 = Float) -> Bool
   | And | Or when same && t1 = Bool -> Bool
   | Div when t1 = Matrix && t2 == Float -> Matrix
   | Mult when t1 = Matrix && t2 == Float -> Matrix
   | Mult when t2 = Matrix && t1 == Float -> Matrix
   | _ -> raise (
      Failure ("illegal binary operator " ^
               string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
               string_of_typ t2 ^ " in " ^ string_of_expr e))
in
   let (t1, e1') = match e1 with
     Literal l when ty = Float -> (Float, SFloatLit (string_of_int l))
   | _ -> expr e1 map

   and (t2, e2') = match e2 with
     Literal l when ty = Float -> (Float, SFloatLit (string_of_int l))
   | _ -> expr e2 map
```

```
      in (ty, SBinop((t1, e1'), op, (t2, e2')))

  | MatAccess(em, e1, e2) -> let (tm, sem) = (expr em map) in
    (match tm with Matrix ->
      let (t1, se1) = expr e1 map
      in  let (t2,se2) = expr e2 map
          in (match t1,t2 with
              Int,Int -> (Float, SMatAccess((tm,sem), (t1, se1), (t2, se2)))
            | _ -> raise ( Failure ("Index is not a integer!")))
      | _ -> raise ( Failure ("matrix access on non-matrix"))
    )

  | MatSlice(m, r1, r2) ->
    let r1', r2' = expr r1 map, expr r2 map in
    let m' = expr m map in
    (Matrix, SMatSlice(m', r1', r2'))
  | Range(r1, r2) -> (match r1, r2 with
        Beg, End -> (Rangety, SRange(SBeg, SEnd))
      | Beg, Ind(i) -> (Rangety, SRange(SBeg, SInd(expr i map)))
      | Ind(i), End -> (Rangety, SRange(SInd(expr i map), SEnd))
      | Ind(i1), Ind(i2) when i1 <= i2 -> (Rangety, SRange(SInd(expr i1 map), SInd(expr i2 map
          )))
      | _ -> raise (Failure "Invalid ranges")
    )
  | CastExpr(ty, e) -> let se = expr e map in (ty, SCastExpr(ty, se))
  | StringAccess(s, i) ->
      let (ts, ses) = expr s map in
      (match ts with String ->
        let (t2,se2) = expr i map in
        (match t2 with
            Int -> (Char, SStringAccess((ts, ses), (t2, se2)))
          | _ -> raise ( Failure ("Index is not an integer!")))
        | _ -> raise ( Failure ("string access on non-string")))
  | Call(fname, args) as call ->
      let fd = find_func fname in
      let param_length = List.length fd.formals in
      if List.length args != param_length then
        raise (Failure ("expecting " ^ string_of_int param_length ^
                        " arguments in " ^ string_of_expr call))
      else let check_call (ft, _) e =
        let (et, e') = expr e map in
        let err = "illegal argument found " ^ string_of_typ et ^
          " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
        in (check_assign ft et err, e')
      in
      let args' = List.map2 check_call fd.formals args
      in (fd.typ, SCall(fname, args'))
  in


let check_function func =
  (* Make sure no formals or locals are void or duplicates *)

  check_binds "formal" func.formals;
  (* check_binds "local" func.locals; *)


  (* Build local symbol table of variables for this function *)
  let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
```

```
                StringMap.empty (global_symbols globals @ func.formals)

in

let check_bool_expr e vars =
  let (t', e') = expr e vars
  and err = "expected Boolean expression in " ^ string_of_expr e
  in if t' != Bool then raise (Failure err) else (t', e')
in

(* Return a semantically-checked statement i.e. containing sexprs *)
let rec check_stmt stmt vars = match stmt with
    Expr e -> (SExpr (expr e vars), vars)
  | If(p, b1, b2) -> (SIf(check_bool_expr p vars, fst (check_stmt b1 vars), fst (check_stmt b2
      vars)), vars)
  | For(e1, e2, e3, st) ->
          (SFor(expr e1 vars, check_bool_expr e2 vars, expr e3 vars, fst(check_stmt st vars)),
              vars)
  | While(p, s) -> (SWhile(check_bool_expr p vars, fst (check_stmt s vars)), vars)
  | Return e -> let (t, e') = expr e vars in
     if t = func.typ then (SReturn (t, e'), vars)
     else raise (
     Failure ("return gives " ^ string_of_typ t ^ " expected " ^
     string_of_typ func.typ ^ " in " ^ string_of_expr e))
  | VarDecl(v) -> (match v with
     Decl(ty, n) -> (SVarDecl(SDecl(ty, n)), StringMap.add n ty vars)
   | DeclInit((ty, n), e) -> (SVarDecl(SDeclInit((ty, n), expr e vars)), StringMap.add n ty
       vars))

       (* A block is correct if each statement is correct and nothing
          follows any Return statement. Nested blocks are flattened. *)
  | Block sl ->
     let bvars = StringMap.empty in
     let rec check_stmt_list stmt_list vars bvars =
       let check s ss bvars =
         let checked = check_stmt s vars in
         let checked_rest = check_stmt_list ss (snd checked) bvars in
         (fst checked :: fst checked_rest, snd checked_rest)
       in
       match stmt_list with
         [Return _ as s] -> ([fst (check_stmt s vars)], bvars)
       | Return _ :: _ -> raise (Failure "nothing may follow a return")
       | VarDecl v as s :: ss ->
         let check_decl ty n = match ty with
           Void -> raise (Failure ("illegal void local variable " ^ n))
         | _ -> (match StringMap.find_opt n bvars with
             Some _ -> raise (Failure ("duplicate local variable " ^ n))
           | None -> ())
         in (match v with
              Decl(ty, n) -> ignore(check_decl ty n); check s ss (StringMap.add n ty bvars) (*
                  (fst checked, StringMap.add n ty (snd checked)) *)
            | DeclInit((ty, n), _) -> ignore(check_decl ty n); check s ss (StringMap.add n ty
                bvars)) (* (fst checked, StringMap.add n ty (snd checked))) *)
       | s :: ss       -> check s ss bvars
       | []            -> ([], bvars)
     in (SBlock(fst (check_stmt_list sl vars bvars)), vars)
  | Continue -> (SContinue, vars)
  | Break -> (SBreak, vars)
```

```
    in (* body of check_function *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      sbody = match fst (check_stmt (Block func.body) symbols) with
        SBlock(sl) -> sl
      | _ -> raise (Failure ("internal error: block didn't become a block?"))
    }
  in

  let check_global glob =
    (* Build local symbol table of variables for this function *)
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
                StringMap.empty (global_symbols globals)
                (* concats two diff types into a list *)

    in
    match glob with
        DeclInit(b, e) -> SDeclInit(b, expr e symbols)
      | Decl(d) -> SDecl(d)
  in (List.map check_global globals, List.map check_function functions)
```

8.2.5 sast.ml

```
(*Semantically Checked AST *)

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SBoolLit of bool
  | SCharLit of char
  | SStringLit of string
  | SFloatLit of string
  | SMatrixLit of float array * (int * int)
  | SMatAccess of sexpr * sexpr * sexpr
  | SMatSlice of sexpr * sexpr * sexpr
  | SStringAccess of sexpr * sexpr
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SCall of string * sexpr list
  | SCastExpr of typ * sexpr
  | SNoexpr
  | SRange of sindex * sindex
  and sindex = SBeg | SEnd | SInd of sexpr

type sbind_decl = typ * string

type sbind_init = sbind_decl * sexpr

type sbind =
   SDecl of sbind_decl
 | SDeclInit of sbind_init

type sstmt =
```

```
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SContinue
  | SBreak
  | SVarDecl of sbind

type sfunc_decl = {
    styp : typ;
    sfname : string;
    sformals : bind_decl list;
    sbody : sstmt list;
  }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)


let string_of_sop = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Elemult -> ".*"
  | Elediv -> "./"

let string_of_suop = function
    Neg -> "-"
  | Not -> "!"

let string_of_styp = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Char -> "char"
  | String -> "str"
  | Float -> "float"
  | Matrix -> "matrix"
  | Rangety -> "range"

let rec string_of_smatrix scalar cols _ =
  "[ " ^ (let rec string_of_elems = function
      [] -> ""
    | h::t -> string_of_sexpr h ^
              (if (List.length t) == 0 then ""
              else if (List.length t) mod cols == 0 then "; "
              else ", ") ^
```

```
            string_of_elems t
  in string_of_elems scalar) ^ " ]"

and string_of_sexpr (t,e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SCharLit(c) -> "'" ^ String.make 1 c ^ "'"
  | SStringLit(s) -> "\"" ^ s ^ "\""
  | SFloatLit(f) -> f
  | SMatrixLit(s, (c, l)) -> let s_array = (Array.map (fun x -> Float, SFloatLit(Printf.sprintf "%g
      " x))) s in string_of_smatrix (Array.to_list s_array) c l
  | SMatAccess(m, c, r) -> string_of_sexpr m ^ "[" ^ string_of_sexpr c ^ "," ^ string_of_sexpr r ^
      "]"
  | SMatSlice(m, r1, r2) -> string_of_sexpr m ^ "[" ^ string_of_sexpr r1 ^ "," ^ string_of_sexpr r2
      ^ "]"
  | SStringAccess(s, i) -> string_of_sexpr s ^ "[" ^ string_of_sexpr i ^ "]"
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_sop o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_suop o ^ string_of_sexpr e
  | SAssign(v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SCastExpr(t, e) -> "(" ^ string_of_styp t ^ ") " ^ string_of_sexpr e
  | SNoexpr -> ""
  | SRange(s1, t1) -> (match s1, t1 with
     SBeg, SEnd -> ":"
    | SBeg, SInd(i) -> ":"^string_of_sexpr i
    | SInd(i), SEnd -> string_of_sexpr i^":"
    | SInd(i1), SInd(i2) -> string_of_sexpr i1^":"^string_of_sexpr i2
    | _ -> raise (Failure "Semant should fail on invalid ranges")
  )
  )^ ")"

(* unsure if should change decl to sdecl*)
let string_of_svdecl = function
    SDecl(decl) -> string_of_styp (fst decl) ^ " " ^ (snd decl) ^ ";\n"
  | SDeclInit(decl, exp) -> string_of_styp (fst decl) ^ " " ^ (snd decl) ^ " = " ^ string_of_sexpr
      exp ^ ";\n"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
     "for (" ^ string_of_sexpr e1 ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
     string_of_sexpr e3 ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
  | SContinue -> "continue;\n"
  | SBreak -> "break;\n"
  | SVarDecl(v) -> string_of_svdecl v

let string_of_simport = function
```

```
  Import(filename) -> "#import <" ^ filename ^ ">\n"

let string_of_sfdecl fdecl =
  string_of_styp fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (vars, funcs) = String.concat "\n" (List.map string_of_svdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

8.2.6 codegen.ml

```
  (* Code generation: translate takes a semantically checked AST and
produces LLVM IR
LLVM tutorial: Make sure to read the OCaml version of the tutorial
http://llvm.org/docs/tutorial/index.html
Detailed documentation on the OCaml LLVM library:
http://llvm.moe/
http://llvm.moe/ocaml/
*)

module L = Llvm
module A = Ast

open Sast
module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context    = L.global_context () in
  let the_module = L.create_module context "YAMML" in
  let i32_t      = L.i32_type context in
  let i64_t      = L.i64_type context in
  let float_t    = L.double_type context in
  let void_t     = L.void_type  context in
  let pointer_t  = L.pointer_type in
  let i1_t       = L.i1_type    context
  and i8_t       = L.i8_type  context in
  let matrix_t   = L.named_struct_type context "matrix_t" in
    L.struct_set_body matrix_t [|L.pointer_type float_t; i32_t; i32_t|] false;

  let ltype_of_typ = function
      A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Char -> i8_t
    | A.Float -> float_t
    | A.Void -> void_t
    | A.String -> pointer_t i8_t
    | A.Matrix -> matrix_t
    | A.Rangety -> raise (Failure "internal error: Rangety not an LLVM type")
  in

  let define_matrix_lit_global(f_array,(r,c)) =
    let lf_array = Array.map (fun f -> L.const_float float_t f) f_array in
    let mat_arr = L.const_array float_t lf_array in
    let mat_def = L.define_global ".mat_array" mat_arr the_module in
    let mat = L.const_gep mat_def [|L.const_int i64_t 0; L.const_int i64_t 0|] in
```

```
  L.const_named_struct matrix_t [|mat; L.const_int i32_t r; L.const_int i32_t c|]
in

(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m b =
    match b with
        SDeclInit((t,n),e) ->
          let rec const_expr ((_, e) : sexpr) = match e with
            SLiteral i -> L.const_int (ltype_of_typ t) i
          | SBoolLit b -> L.const_int (ltype_of_typ t) (if b then 1 else 0)
          | SCharLit c -> L.const_int (ltype_of_typ t) (Char.code c)
          | SStringLit s -> let alloc = L.define_global ".str" (L.const_stringz context s)
              the_module in
                        L.const_in_bounds_gep alloc [|L.const_int i64_t 0; L.const_int i64_t 0|]
          | SMatrixLit (f_array, (r,c)) -> define_matrix_lit_global (f_array, (r, c))
          | SFloatLit l -> L.const_float_of_string float_t l
          | SCastExpr (ct, ((et, _) as e)) ->
            if et = ct then const_expr e else let op = (match et,ct with
              A.Int,A.Float
            | A.Char,A.Float
            | A.Bool,A.Float -> L.const_sitofp
            | A.Float,A.Int
            | A.Float,A.Char
            | A.Float,A.Bool -> L.const_fptosi
            | A.Int,A.Char
            | A.Int,A.Bool
            | A.Char,A.Bool -> L.const_trunc
            | A.Bool, A.Char
            | A.Bool, A.Int
            | A.Char, A.Int -> L.const_sext
            | _,_ -> raise (Failure ("internal error: semant should have rejected cast " ^ A.
                string_of_typ et ^ " -> " ^ A.string_of_typ ct)))
            in op (const_expr e) (ltype_of_typ ct)
          | SNoexpr -> raise (Failure "empty global initializer")
          | SBinop ((A.Float,_ ) as e1, op, e2) ->
            let e1' = match e1 with
              (A.Float, _) -> const_expr e1
            | _ -> L.const_sitofp (const_expr e1) float_t
            and e2' = match e2 with
              (A.Float,_) -> const_expr e2
            | _ -> L.const_sitofp (const_expr e2) float_t in
            (match op with
              A.Add    -> L.const_fadd
            | A.Sub    -> L.const_fsub
            | A.Mult   -> L.const_fmul
            | A.Div    -> L.const_fdiv
            | A.Equal  -> L.const_fcmp L.Fcmp.Oeq
            | A.Neq    -> L.const_fcmp L.Fcmp.One
            | A.Less   -> L.const_fcmp L.Fcmp.Olt
            | A.Leq    -> L.const_fcmp L.Fcmp.Ole
            | A.Greater -> L.const_fcmp L.Fcmp.Ogt
            | A.Geq    -> L.const_fcmp L.Fcmp.Oge
            | A.And | A.Or -> raise (Failure "internal error: semant should have rejected and/or
                on float")
            | A.Elemult | A.Elediv -> raise (Failure "internal error: semant should have rejected
                Elemult/Elediv on float")
            ) e1' e2'
          | SBinop (e1, op, e2) ->
```

```
            let e1' = const_expr e1
            and e2' = const_expr e2 in
            (match op with
              A.Add    -> L.const_add
            | A.Sub    -> L.const_sub
            | A.Mult   -> L.const_mul
            | A.Div    -> L.const_sdiv
            | A.And    -> L.const_and
            | A.Or     -> L.const_or
            | A.Equal  -> L.const_icmp L.Icmp.Eq
            | A.Neq    -> L.const_icmp L.Icmp.Ne
            | A.Less   -> L.const_icmp L.Icmp.Slt
            | A.Leq    -> L.const_icmp L.Icmp.Sle
            | A.Greater -> L.const_icmp L.Icmp.Sgt
            | A.Geq    -> L.const_icmp L.Icmp.Sge
            | A.Elemult | A.Elediv -> raise (Failure "internal error: semant should have rejected
                Elemult/Elediv on int")
            ) e1' e2'
          | SUnop(op, ((t, _) as e)) ->
            let e' = const_expr e in
            (match op with
              A.Neg when t = A.Float -> L.const_fneg
            | A.Neg                 -> L.const_neg
            | A.Not                 -> L.const_not) e'
          | SMatAccess(_,_,_)
          | SMatSlice(_,_,_)
          | SStringAccess(_,_)
          | SId(_)
          | SAssign(_,_)
          | SCall(_,_)
          | SRange(_,_) -> raise (Failure "non-constant global initializer")
        in StringMap.add n (L.define_global n (const_expr e) the_module) m
     | SDecl(t,n) ->
        let init = L.const_int (ltype_of_typ t) 0
        in StringMap.add n (L.define_global n init the_module) m
   in
  List.fold_left global_var StringMap.empty globals in

let printf_t : L.lltype =
   L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

let ssum_t : L.lltype =
  L.function_type float_t [| L.pointer_type float_t; i32_t; i32_t|] in
let ssum_func : L.llvalue = L.declare_function "ssum" ssum_t the_module in

let smean_t : L.lltype =
  L.function_type float_t [| L.pointer_type float_t; i32_t; i32_t|] in
let smean_func : L.llvalue = L.declare_function "smean" smean_t the_module in

let strans_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type float_t;
     i32_t; i32_t|] in
let strans_func : L.llvalue = L.declare_function "strans" strans_t the_module in

let sfilter2D_t : L.lltype = L.function_type void_t [| L.pointer_type matrix_t; L.pointer_type
    matrix_t; i32_t; L.pointer_type matrix_t|] in
let sfilter2D_func : L.llvalue = L.declare_function "sfilter2D" sfilter2D_t the_module in

let simread_t : L.lltype = L.function_type void_t [|pointer_t i8_t; pointer_t matrix_t|] in
```

```
let simread_func : L.llvalue = L.declare_function "simread" simread_t the_module in

let simwrite_t : L.lltype = L.function_type i1_t [|pointer_t i8_t; L.pointer_type matrix_t|] in
let simwrite_func : L.llvalue = L.declare_function "simwrite" simwrite_t the_module in

let sconsmul_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type
    float_t; i32_t; i32_t; float_t|] in
let sconsmul_func : L.llvalue = L.declare_function "sconsmul" sconsmul_t the_module in

let sconsdiv_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type
    float_t; i32_t; i32_t; float_t|] in
let sconsdiv_func : L.llvalue = L.declare_function "sconsdiv" sconsdiv_t the_module in

let smatmul_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type float_t
    ; L.pointer_type float_t; i32_t; i32_t; i32_t; i32_t|] in
let smatmul_func : L.llvalue = L.declare_function "smatmul" smatmul_t the_module in

let selemul_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type float_t
    ; L.pointer_type float_t; i32_t; i32_t; i32_t; i32_t|] in
let selemul_func : L.llvalue = L.declare_function "selemul" selemul_t the_module in
let selediv_t : L.lltype = L.function_type void_t [| L.pointer_type float_t; L.pointer_type float_t
    ; L.pointer_type float_t; i32_t; i32_t; i32_t; i32_t|] in
let selediv_func : L.llvalue = L.declare_function "selediv" selediv_t the_module in
let sprintm_t : L.lltype = L.function_type i32_t [| L.pointer_type float_t; i32_t; i32_t|] in
let sprintm_func : L.llvalue = L.declare_function "sprintm" sprintm_t the_module in

let smatslice_t : L.lltype = L.function_type void_t [| L.pointer_type matrix_t; i32_t; i32_t; i32_t
    ; i32_t; i32_t; i32_t; L.pointer_type matrix_t|] in
let smatslice_func : L.llvalue = L.declare_function "smatslice" smatslice_t the_module in

let sinvert_t : L.lltype = L.function_type void_t [| L.pointer_type matrix_t; L.pointer_type
    matrix_t|] in
let sinvert_func : L.llvalue = L.declare_function "sinvert" sinvert_t the_module in

let function_decls =
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
          in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
          StringMap.add name (L.define_function name ftype the_module, fdecl) m in
     List.fold_left function_decl StringMap.empty functions in

  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.sfname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    let string_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
    let float_format_str = L.build_global_stringptr "%g\n" "fmt" builder in
    let char_format_str = L.build_global_stringptr "%c\n" "fmt" builder in


    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup map n = try StringMap.find n map
                  with Not_found -> try StringMap.find n global_vars
                  with Not_found -> raise (Failure ("Not defined: " ^ n))
    in
```

```
let build_matrix_lit(f_array,(r,c)) builder =
  let mat = L.build_array_malloc float_t (L.const_int i32_t (r*c)) "system_mat" builder in
  (for x = 0 to (r*c-1) do
    let element_ptr = L.build_gep mat [|(L.const_int i32_t x)|] "element_ptr" builder in
    ignore(L.build_store (L.const_float float_t f_array.(x)) element_ptr builder)
  done);
  let m = L.build_alloca matrix_t "m" builder in
  let m_mat = L.build_struct_gep m 0 "m_mat" builder in ignore(L.build_store mat m_mat builder);
  let m_r = L.build_struct_gep m 1 "m_r" builder in ignore(L.build_store (L.const_int i32_t r)
      m_r builder);
  let m_c = L.build_struct_gep m 2 "m_c" builder in ignore(L.build_store (L.const_int i32_t c)
      m_c builder);
  m
in

let build_default_mat (r,c) builder =
  let mat = L.build_array_malloc float_t (L.build_mul r c "tmp" builder) "system_mat" builder in
  let m = L.build_alloca matrix_t "m" builder in
  let m_mat = L.build_struct_gep m 0 "m_mat" builder in ignore(L.build_store mat m_mat builder);
  let m_r = L.build_struct_gep m 1 "m_r" builder in ignore(L.build_store r m_r builder);
  let m_c = L.build_struct_gep m 2 "m_c" builder in ignore(L.build_store c m_c builder);
  m
in
let build_empty_mat builder =
  let m = L.build_alloca matrix_t "m" builder in
  let m_r = L.build_struct_gep m 1 "m_r" builder in ignore(L.build_store (L.const_int i32_t 0)
      m_r builder);
  let m_c = L.build_struct_gep m 2 "m_c" builder in ignore(L.build_store (L.const_int i32_t 0)
      m_c builder);
  m
in
let build_matrix_elem_ptr s i j builder =
  let mat = L.build_load (L.build_struct_gep s 0 "m_mat" builder) "mat" builder in
  let c = L.build_load (L.build_struct_gep s 2 "m_c" builder) "c_mat" builder in
  let index = L.build_add j (L.build_mul i c "tmp" builder) "index" builder in
  L.build_gep mat [|index|] "element_ptr_ptr" builder
in

let mat_float_mult m num builder =
  let s' = L.build_load (L.build_struct_gep m 0 "m_mat" builder) "m" builder in
  let r = L.build_load (L.build_struct_gep m 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep m 2 "m_c" builder) "c_mat" builder in
  let res_mat = build_default_mat (r,c) builder in
  let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
  ignore(L.build_call sconsmul_func [|s';res;r;c; num|] "" builder);
  res_mat
in

let mat_float_div m num builder =
  let s' = L.build_load (L.build_struct_gep m 0 "m_mat" builder) "m" builder in
  let r = L.build_load (L.build_struct_gep m 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep m 2 "m_c" builder) "c_mat" builder in
  let res_mat = build_default_mat (r,c) builder in
  let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
  ignore(L.build_call sconsdiv_func [|s';res;r;c; num|] "" builder);
  res_mat
in
(*elementwise matrix multiplication operator*)
```

```
  let mat_elemul_operation m1 m2 builder =
    let s1' = L.build_load (L.build_struct_gep m1 0 "m_mat" builder) "m" builder in
    let s2' = L.build_load (L.build_struct_gep m2 0 "m_mat" builder) "m" builder in
    let r1 = L.build_load (L.build_struct_gep m1 1 "m_r" builder) "r_mat" builder in
    let c1 = L.build_load (L.build_struct_gep m1 2 "m_c" builder) "c_mat" builder in
    let r2 = L.build_load (L.build_struct_gep m2 1 "m_r" builder) "r_mat" builder in
    let c2 = L.build_load (L.build_struct_gep m2 2 "m_c" builder) "c_mat" builder in

    let res_mat = build_default_mat (r1,c1) builder in
    let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
    ignore(L.build_call selemul_func [|s1'; s2'; res; r1; c1; r2; c2|] "" builder);
    res_mat
in
  let mat_elediv_operation m1 m2 builder =
    let s1' = L.build_load (L.build_struct_gep m1 0 "m_mat" builder) "m" builder in
    let s2' = L.build_load (L.build_struct_gep m2 0 "m_mat" builder) "m" builder in
    let r1 = L.build_load (L.build_struct_gep m1 1 "m_r" builder) "r_mat" builder in
    let c1 = L.build_load (L.build_struct_gep m1 2 "m_c" builder) "c_mat" builder in
    let r2 = L.build_load (L.build_struct_gep m2 1 "m_r" builder) "r_mat" builder in
    let c2 = L.build_load (L.build_struct_gep m2 2 "m_c" builder) "c_mat" builder in

    let res_mat = build_default_mat (r1,c1) builder in
    let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
    ignore(L.build_call selediv_func [|s1'; s2'; res; r1; c1; r2; c2|] "" builder);
    res_mat
in
  let mat_matmul_operation m1 m2 builder =
    let s1' = L.build_load (L.build_struct_gep m1 0 "m_mat" builder) "m" builder in
    let s2' = L.build_load (L.build_struct_gep m2 0 "m_mat" builder) "m" builder in
    let r1 = L.build_load (L.build_struct_gep m1 1 "m_r" builder) "r_mat" builder in
    let c1 = L.build_load (L.build_struct_gep m1 2 "m_c" builder) "c_mat" builder in
    let r2 = L.build_load (L.build_struct_gep m2 1 "m_r" builder) "r_mat" builder in
    let c2 = L.build_load (L.build_struct_gep m2 2 "m_c" builder) "c_mat" builder in
    let res_mat = build_default_mat (r1,c2) builder in
    let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
    ignore(L.build_call smatmul_func [|s1'; s2'; res; r1; c1; r2; c2|] "" builder);
    res_mat
in
    let rec expr_ptr builder map se =
      let e = snd se in match e with
        SId s -> lookup map s
      | SMatrixLit(f_array, (r,c)) -> build_matrix_lit (f_array, (r,c)) builder
      | SBinop (e1, op, ((A.Matrix,_ ) as e2))
      | SBinop ((A.Matrix,_ ) as e1, op, e2) -> (
        match (e1, op, e2) with
          ((A.Matrix,_), A.Div, (A.Float,_)) ->
            let e1' = expr_ptr builder map e1 in
            let e2' = expr builder map e2 in mat_float_div e1' e2' builder
        | ((A.Matrix,_), A.Mult, (A.Float,_)) ->
            let e1' = expr_ptr builder map e1 in
            let e2' = expr builder map e2 in mat_float_mult e1' e2' builder
        | ((A.Float,_), A.Mult, (A.Matrix,_)) ->
        let e1' = expr builder map e1 in
        let e2' = expr_ptr builder map e2 in mat_float_mult e2' e1' builder
        | ((A.Matrix,_), A.Mult, (A.Matrix,_)) ->
            let e1' = expr_ptr builder map e1 in
            let e2' = expr_ptr builder map e2 in mat_matmul_operation e1' e2' builder
        | ((A.Matrix,_), A.Elemult, (A.Matrix,_)) ->
```

```
      let e1' = expr_ptr builder map e1 in
      let e2' = expr_ptr builder map e2 in mat_elemul_operation e1' e2' builder
  | ((A.Matrix,_), A.Elediv, (A.Matrix,_)) ->
      let e1' = expr_ptr builder map e1 in
      let e2' = expr_ptr builder map e2 in mat_elediv_operation e1' e2' builder
  | _ -> raise (Failure "Not supported yet"))
| SMatSlice (sem, e1, e2) ->
  let s' = expr_ptr builder map sem in
  let r = L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder in
  let range_ind max = function
    SBeg -> L.const_int i32_t 0
  | SEnd -> L.build_sub max (L.const_int i32_t 1) ".tmp_sub" builder
  | SInd i -> expr builder map i
  in
  let range_inds max = function
    SRange(a,b) -> (range_ind max a),(range_ind max b)
  | _ -> let ind = expr builder map e1 in ind,ind
  in
  let i,j = range_inds r (snd e1) in
  let k,l = range_inds c (snd e2) in
  let res_mat = build_empty_mat builder in
  ignore(L.build_call smatslice_func [|s'; r; c; i; j; k; l; res_mat|] "" builder);
  res_mat
| SCall ("trans", [e]) ->
  let s' = expr_ptr builder map e in
  let mat = L.build_load (L.build_struct_gep s' 0 "m_mat" builder) "mat" builder in
  let r = L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder in
  let res_mat = build_default_mat (c,r) builder in
  let res = L.build_load (L.build_struct_gep res_mat 0 "m_mat" builder) "mat" builder in
  ignore (L.build_call strans_func [|mat;res;r;c|] "" builder );
  res_mat

| SCall ("empty", [r;c]) ->
  let r = expr builder map r in
  let c = expr builder map c in
  let res_mat = build_default_mat (r,c) builder in
  res_mat
| SCall ("imread", [f]) ->
  let f' = expr builder map f in
  let res_mat = build_empty_mat builder in
  ignore(L.build_call simread_func [|f'; res_mat|] "" builder);
  res_mat
| SCall (f, _) ->
    let (_, fdecl) = StringMap.find f function_decls in
    let tmp_var = L.build_alloca (ltype_of_typ fdecl.styp) ".ret" builder
    in ignore (L.build_store (expr builder map se) tmp_var builder);
    tmp_var
| _ -> raise (Failure ("internal error: need alloc ptr for expression of type " ^ A.
    string_of_typ (fst se)))
and expr builder map se =
  let e = snd se in match e with
    SLiteral i -> L.const_int i32_t i
  | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
  | SCharLit c -> L.const_int i8_t (Char.code c)
  | SMatrixLit (_,_) -> L.build_load (expr_ptr builder map se) ".matlit" builder
  | SId s      -> L.build_load (expr_ptr builder map se) s builder
  | SStringLit s -> L.build_global_stringptr s ".strlit" builder
```

```
| SFloatLit l -> L.const_float_of_string float_t l
| SCastExpr (ct, ((et, _) as e)) ->
  let e' = expr builder map e in
  if et = ct then e' else let op = (match et,ct with
    A.Int,A.Float
  | A.Char,A.Float
  | A.Bool,A.Float -> L.build_sitofp
  | A.Float,A.Int
  | A.Float,A.Char
  | A.Float,A.Bool -> L.build_fptosi
  | A.Int,A.Char
  | A.Int,A.Bool
  | A.Char,A.Bool -> L.build_trunc
  | A.Bool, A.Char
  | A.Bool, A.Int
  | A.Char, A.Int -> L.build_sext
  | _,_ -> raise (Failure ("internal error: semant should have rejected cast " ^ A.
      string_of_typ et ^ " -> " ^ A.string_of_typ ct)))
  in op e' (ltype_of_typ ct) ".cast_op" builder
| SNoexpr    -> L.const_int i32_t 0
| SBinop ((A.Matrix,_ ), _, _)
| SBinop (_, _, ((A.Matrix,_ ))) -> L.build_load (expr_ptr builder map se) ".matop" builder
| SBinop ((A.Float,_ ) as e1, op, e2) ->
  let e1' = expr builder map e1
  and e2' = expr builder map e2
  in (match(e1, e2) with
  | _ -> (match op with
      A.Add    -> L.build_fadd
    | A.Sub    -> L.build_fsub
    | A.Mult   -> L.build_fmul
    | A.Div    -> L.build_fdiv
    | A.Equal  -> L.build_fcmp L.Fcmp.Oeq
    | A.Neq    -> L.build_fcmp L.Fcmp.One
    | A.Less   -> L.build_fcmp L.Fcmp.Olt
    | A.Leq    -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq    -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
      raise (Failure "internal error: semant should have rejected and/or on float")
    | A.Elemult | A.Elediv ->
      raise (Failure "internal error: semant should have rejected element mult/div on float
          ")
  ) e1' e2' "tmp" builder)
| SBinop (e1, op, e2) ->
  let e1' = expr builder map e1
  and e2' = expr builder map e2 in
  (match op with
    A.Add    -> L.build_add
  | A.Sub    -> L.build_sub
  | A.Mult   -> L.build_mul
  | A.Div    -> L.build_sdiv
  | A.And    -> L.build_and
  | A.Or     -> L.build_or
  | A.Equal  -> L.build_icmp L.Icmp.Eq
  | A.Neq    -> L.build_icmp L.Icmp.Ne
  | A.Less   -> L.build_icmp L.Icmp.Slt
  | A.Leq    -> L.build_icmp L.Icmp.Sle
  | A.Greater -> L.build_icmp L.Icmp.Sgt
  | A.Geq    -> L.build_icmp L.Icmp.Sge
```

```
  | A.Elemult | A.Elediv ->
    raise (Failure "internal error: semant should have rejected element mult/div on int")
  ) e1' e2' "tmp" builder


| SMatSlice (_,_,_) -> L.build_load (expr_ptr builder map se) ".matslice" builder

| SMatAccess (sem, e1, e2) ->
    let idx =
      let e1' = expr builder map e1
      and e2' = expr builder map e2
      and s' = expr_ptr builder map sem in
      build_matrix_elem_ptr s' e1' e2' builder
    in
    L.build_load idx "element_ptr" builder

| SStringAccess (s, i) ->
    let idx =
      let s' = expr builder map s in
      let index = expr builder map i in
      L.build_gep s' [|index|] ".char_ptr" builder
    in
    L.build_load idx ".char" builder
| SUnop(op, ((t, _) as e)) ->
    let e' = expr builder map e in
  (match op with
    A.Neg when t = A.Float -> L.build_fneg
  | A.Neg                  -> L.build_neg
  | A.Not                  -> L.build_not) e' "tmp" builder
| SAssign ((_, s), e) -> let e' = expr builder map e in
  let dst = (match s with
    SId v -> lookup map v
  | SMatAccess (sem, e1, e2) ->
      let e1' = expr builder map e1
      and e2' = expr builder map e2
      and s' = expr_ptr builder map sem in
      build_matrix_elem_ptr s' e1' e2' builder
  | SStringAccess (s, i) ->
      let s' = expr builder map s in
      let index = expr builder map i in
      L.build_gep s' [|index|] ".char_ptr" builder
  | _ -> raise (Failure "internal error: semant should have rejected assignment on invalid
      expression"))
  in ignore(L.build_store e' dst builder); e'
| SCall ("height", [sem]) ->
    let s' = expr_ptr builder map sem in
    L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder
| SCall ("width", [sem]) ->
    let s' = expr_ptr builder map sem in
      L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder
| SCall ("printf", [e]) ->
  L.build_call printf_func [| float_format_str ; (expr builder map e) |]
    "printf" builder
| SCall ("printb", [e]) ->
  L.build_call printf_func [| int_format_str ; (expr builder map e) |]
    "printf" builder
| SCall ("print", [e]) ->
  L.build_call printf_func [| int_format_str ; (expr builder map e) |]
    "printf" builder
```

```
| SCall ("prints", [e]) ->
  L.build_call printf_func [| string_format_str ; (expr builder map e) |]
    "printf" builder
| SCall ("printc", [e]) ->
  L.build_call printf_func [| char_format_str ; (expr builder map e) |]
    "printf" builder
| SCall ("printm", [sem]) ->
  let s' = expr_ptr builder map sem in
  let mat = L.build_load (L.build_struct_gep s' 0 "m_mat" builder) "mat_mat" builder in
  let r = L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder in
  L.build_call sprintm_func [|mat; r; c|] "" builder
| SCall ("sum", [sem]) ->
  let s' = expr_ptr builder map sem in
  let mat = L.build_load (L.build_struct_gep s' 0 "m_mat" builder) "mat_mat" builder in
  let r = L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder in
  L.build_call ssum_func [|mat; r; c|] "sum" builder
| SCall ("mean", [sem]) ->
  let s' = expr_ptr builder map sem in
  let mat = L.build_load (L.build_struct_gep s' 0 "m_mat" builder) "mat_mat" builder in
  let r = L.build_load (L.build_struct_gep s' 1 "m_r" builder) "r_mat" builder in
  let c = L.build_load (L.build_struct_gep s' 2 "m_c" builder) "c_mat" builder in
  L.build_call smean_func [|mat; r; c|] "mean" builder

| SCall ("trans", _) -> L.build_load (expr_ptr builder map se) ".mattrans" builder
| SCall ("empty", _) -> L.build_load (expr_ptr builder map se) ".matempty" builder

| SCall ("filter2D", [src; dst; ddepth; kern]) ->
  let ss' = expr_ptr builder map src in
  let ds' = expr_ptr builder map dst in
  let dd = expr builder map ddepth in
  let ks' = expr_ptr builder map kern in
  L.build_call sfilter2D_func [|ss'; ds'; dd; ks'|] "" builder

| SCall ("imread", _) -> L.build_load (expr_ptr builder map se) ".matret" builder

| SCall ("imwrite", [f; m]) ->
  let f' = expr builder map f in
  let m' = expr_ptr builder map m in
  L.build_call simwrite_func [|f'; m'|] ".matret" builder

| SCall ("invert", [src; dst;]) ->
  let ss' = expr_ptr builder map src in
  let ds' = expr_ptr builder map dst in
  L.build_call sinvert_func [|ss'; ds'|] "" builder

| SCall (f, args) ->

  let (fdef, fdecl) = StringMap.find f function_decls in
  let llargs = List.rev (List.map (expr builder map) (List.rev args)) in
  let result = (match fdecl.styp with
                A.Void -> ""
              | _ -> f ^ "_result") in
  L.build_call fdef (Array.of_list llargs) result builder
| SRange(_,_) -> raise (Failure "internal error: SRange should not reach codegen as
    expression")
in
```

```
(* Construct the function's "locals": formal arguments and locally
  declared variables. Allocate each on the stack, initialize their
  value, if appropriate, and remember their values in the "locals" map *)
let formal_vars =
  let add_formal m (t, n) p =
    L.set_value_name n p;
    let formal = L.build_alloca (ltype_of_typ t) n builder in
         ignore (L.build_store p formal builder);
    StringMap.add n formal m
  (* Allocate space for any locally declared variables and add the
    * resulting registers to our map *)
  in List.fold_left2 add_formal StringMap.empty fdecl.sformals
    (Array.to_list (L.params the_function))

  in

  let add_terminal builder instr =
       match L.block_terminator (L.insertion_block builder) with
    Some _ -> ()
  | None -> ignore (instr builder) in

let rec stmt (builder, vars, w_bb, wm_bb) = function
    SBlock sl ->
      let fold_stmt (builder, vars) = stmt (builder, vars, w_bb, wm_bb) in
      (fst (List.fold_left fold_stmt (builder, vars) sl), vars)
  | SExpr e -> ignore (expr builder vars e); (builder, vars)
  | SVarDecl vd -> (builder, match vd with
    | SDeclInit((t,n),e) ->
        let e' = expr builder vars e
        in L.set_value_name n e';
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in ignore (L.build_store e' local_var builder);
        StringMap.add n local_var vars
    | SDecl(t,n) ->
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in StringMap.add n local_var vars)
  | SIf (predicate, then_stmt, else_stmt) ->
    let bool_val = expr builder vars predicate in
    let merge_bb = L.append_block context "merge" the_function in
         let build_br_merge = L.build_br merge_bb in (* partial function *)

    let then_bb = L.append_block context "then" the_function in
    add_terminal (fst (stmt ((L.builder_at_end context then_bb), vars, w_bb, wm_bb) then_stmt)
        )
      build_br_merge;

    let else_bb = L.append_block context "else" the_function in
    add_terminal (fst (stmt ((L.builder_at_end context else_bb), vars, w_bb, wm_bb) else_stmt)
        )
      build_br_merge;

    ignore(L.build_cond_br bool_val then_bb else_bb builder);
    (L.builder_at_end context merge_bb, vars)
  | SWhile (predicate, body) ->
    let pred_bb = L.append_block context "while" the_function in
    ignore(L.build_br pred_bb builder);

    let merge_bb = L.append_block context "merge" the_function in
    let body_bb = L.append_block context "while_body" the_function in
```

```
        add_terminal (fst (stmt ((L.builder_at_end context body_bb), vars, Some(body_bb), Some(
            merge_bb)) body))
          (L.build_br pred_bb);

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder vars predicate in

        ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
        (L.builder_at_end context merge_bb, vars)

      (* Implement for loops as while loops *)
      | SFor (e1, e2, e3, body) -> stmt (builder, vars, w_bb, wm_bb)
        ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )

      | SBreak ->
         let while_merge_bb = (match wm_bb with
             Some(bb) -> bb
           | None -> raise (Failure "break statement not in while loop")
         ) in
         ignore(L.build_br while_merge_bb builder);
         (builder, vars)
      | SContinue ->
         let while_body_bb = (match w_bb with
             Some(bb) -> bb
           | None -> raise (Failure "break statement not in while loop")
         ) in
         ignore(L.build_br while_body_bb builder);
         (builder, vars)

      | SReturn e -> ignore(match fdecl.styp with
            (* Special "return nothing" instr *)
            A.Void -> L.build_ret_void builder
            (* Build return statement *)
          | _ -> L.build_ret (expr builder vars e) builder ); (builder, vars)
    in
    let builder = fst(stmt (builder, formal_vars, None, None) (SBlock fdecl.sbody)) in

    add_terminal builder (match fdecl.styp with
        A.Void -> L.build_ret_void
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in
  List.iter build_function_body functions;
 the_module
```

8.2.7 yamml.ml

```
(* Top-level of the YAMML compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

module StringMap = Map.Make(String)

type action = Ast | Sast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
```

```
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./yamml.native [-a|-l|-c] [file.yamml]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf channel = Lexing.from_channel !channel in
  let rec ast channel =
    let (imports, g, f) = Parser.program Scanner.token (lexbuf channel) in
    let import (gl, fu) im =
      let ichannel = ref (open_in (match im with Ast.Import f -> f)) in
      let iast = ast ichannel in
      (fst iast @ gl, snd iast @ fu)
    in
    List.fold_left import (g, f) imports
  in
  match !action with
    Ast -> print_string (Ast.string_of_program (ast channel))
  | _ -> let sast = Semant.check (ast channel) in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)
```

8.2.8 io.cpp

```
//============================================================================
// Name        : load.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Ansi-style
//============================================================================

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <iostream>
#include <string>
#include <cmath>
#include <exception>
using namespace std;

struct MatMulException : public exception {
   const char * what () const throw () {
       return "Unable to multiply matrices due to incorrect dimensions";
   }
};

class AccessException : public exception
{
    const char *what() const throw(){
```

```
        return "Attempted to access an index that is out of range";
    }
};

struct MatrixElementWiseException : public exception
{
    const char *what() const throw()
    {
        return "Unable to perform element-wise operation between matrices of different dimensions";
    }
};

typedef struct
{
    double *arr;
    int32_t r;
    int32_t c;
} matrix_t;

extern "C" int sprintm(double *indata, int r, int c)
{
    putchar('[');
    for (int i = 0; i < r; ++i){
        for (int j = 0; j < c; ++j){
            if (i > 0 && j == 0){
                putchar(' ');
            }
            printf("%.1f", indata[i * c + j]);
            if (j < c - 1){
                putchar('\t');
            }
        }
        if(i < r-1){
            putchar('\n');
        }
    }
    putchar(']');
    putchar('\n');

    return 0;
}

extern "C" void strans(double *indata, double *outdata, int r, int c)
{
    for (int i = 0; i < r; ++i)
    {
        for (int j = 0; j < c; ++j)
        {
            outdata[j * r + i] = indata[i * c + j];
        }
    }
    return;
}

extern "C" bool check_range(int r, int c, int x, int y)
{
    if (x >= 0 && x < r && y >= 0 && y < c)
    {
        return true;
```

```
    }
    else
    {
        return false;
    }
}

extern "C" int saccess(double *indata, int r, int c, int i, int j)
{
    int idx;
    if (check_range(r, c, i, j))
    {
        idx = (i * c) + j;
    }
    else{
        throw AccessException();
    }
    return idx;
}

extern "C" double ssum(double *indata, int r, int c)
{
    double total = 0.0;
    for (int i = 0; i < r*c; ++i)
    {
        total += indata[i];

    }
    return total;
}

extern "C" void sconsmul(double *indata, double *outdata, int r, int c, double cc)
{

    for (int i = 0; i < r * c; ++i)
    {
        outdata[i] = cc * indata[i];
    }
}

extern "C" void sconsdiv(double *indata, double *outdata, int r, int c, double cc)
{

    for (int i = 0; i < r * c; ++i)
    {
        outdata[i] = indata[i] / cc;
    }
}

extern "C" void smatmul(double *indata1, double *indata2, double *outdata, int r1, int c1, int r2,
    int c2)
{

    if (c1 != r2) {
        throw MatMulException();
    }
    else {

        for (int i = 0; i < r1; ++i)
```

```
        {
            for (int j = 0; j < c2; ++j)
            {
                int idx = i * c2 + j;
                float tmp_sum = 0.0;
                int x = i * c1;
                int y = j;
                while (y < r2 * c2)
                {
                    tmp_sum += indata1[x] * indata2[y];
                    x = x + 1;
                    y = y + c2;
                }
                outdata[idx] = tmp_sum;
            }
        }
        return;
    }
}

extern "C" void smatslice(matrix_t *inmat, int r, int c, int i, int j, int k, int l, matrix_t *
    outmat)
{
    if (check_range(r, c, i, k) && check_range(r, c, j, l))
    {
        int outr = outmat->r = j - i + 1, outc = outmat->c = l - k + 1;
        double *indata = inmat->arr,
            *outdata = outmat->arr = (double *)malloc((outr * outc) * sizeof(double));
        int x, y;

        assert(outdata != NULL);

        for (x = i; x <= j; ++x)
            for (y = k; y <= l; ++y)
                outdata[(x - i) * outc + (y - k)] = indata[x * inmat->c + y];
    }
    else{
        throw AccessException();
    }
}

extern "C" int smateleassign(double *indata1, int r, int c, int i, int j, float ele)
{
    int idx;
    indata1[i * c + j] = ele;
    if (check_range(r, c, i, j))
    {
        idx = (i * c) + j;
        indata1[i * c + j] = ele;
    }
    else
    {
        throw AccessException();
    }
    return idx;
}

extern "C" void selemul(double *indata1, double *indata2, double *outdata, int r1, int c1, int r2,
    int c2)
```

```
{
    if (r1 == r2 and c1 == c2){
        for (int i = 0; i < r1*c1; ++i)
        {
            outdata[i] = indata1[i] * indata2[i];
        }
    }
    else
    {
        throw MatrixElementWiseException();
    }
}

extern "C" void selediv(double *indata1, double *indata2, double *outdata, int r1, int c1, int r2,
    int c2)
{

    if (r1 == r2 and c1 == c2)
    {
        for (int i = 0; i < r1 * c1; ++i)
        {
            outdata[i] = indata1[i] / indata2[i];
        }
    }
    else
    {
        throw MatrixElementWiseException();
    }
}

extern "C" double smean(double *indata, int r, int c)
{
    return ssum(indata, r, c) / (r * c);
}
```

8.2.9 opencv.cpp

```
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/imgcodecs.hpp>

using namespace cv;

typedef struct {
        double *arr;
        int32_t r;
        int32_t c;
} matrix_t;

#define MAT(mat) \
        Mat(mat->r, mat->c, CV_64F, mat->arr, sizeof(double) * mat->c)

static inline matrix_t YMAT_1B(Mat mat)
{
        double *arr = (double *)malloc((mat.rows * mat.cols) * sizeof(double));
        matrix_t ymat = { .arr = arr, .r = mat.rows, .c = mat.cols };
        int i;
```

```
	assert(ymat.arr != NULL);
	for (i = 0; i < ymat.r * ymat.c; ++i)
		ymat.arr[i] = (double)((unsigned char *)mat.data)[i];

	return ymat;
}

extern "C" void sfilter2D(matrix_t *src, matrix_t *dst, int ddepth, matrix_t *kernel)
{
	filter2D(MAT(src), MAT(dst), ddepth, MAT(kernel));
}

extern "C" void sinvert(matrix_t *src, matrix_t *dst)
{
	invert(MAT(src),MAT(dst),DECOMP_LU);
}

extern "C" void simread(const char *filename, matrix_t *dst)
{
	Mat mat = imread(filename, IMREAD_GRAYSCALE);
	*dst = YMAT_1B(mat);
}

extern "C" bool simwrite(const char *filename, matrix_t *img)
{
	return imwrite(filename, MAT(img));
}
```

8.3 Standard Library

We did not have too much time to write any other cool standard library functions but as a proof of concept we wrote matrix power in yamml as a standard library

8.3.1 stdlib.yamml

```
matrix pow(matrix m, int exp)
{
matrix res = m;
int i;
for (i = 0; i < exp - 1; i = i + 1)
res = res * m;
return res;
}
```

8.4 Tests

8.4.1 fail-helloworld.yamml

```
int main() {
print('c');
}
```

8.4.2 fail-localoutofscope.yamml

```
int main()
{
```

```
int x = 5;
{
x = x + 1;
int y;
y = 6;
z = x + y;
}
print(z);
}
```

8.4.3 fail-nomain.yamml

This section is deliberately empty to indicate the absence of a main function.

8.4.4 fail-return1.yamml

```
int main(){
return true; /* should return int */
}
```

8.4.5 fail-return2.yamml

```
int main(){
print("hi");
}
```

8.4.6 test-add1.yamml

```
int main()
{
  print(1+1);
}
```

8.4.7 test-addfloat1.yamml

```
int main()
{
  printf(1.0+1.1);
}
```

8.4.8 test-addfloatint1.yamml

```
int main()
{
  printf((1+1.1));
}
```

8.4.9 test-anothermatrix.yamml

```
    matrix myfunc()
    {
    return [ 0.0, 2.0, 5.1 ];
    }

    int main() {
        matrix M = [1.0, 1.1];
        printf(M[0,1]);
        printf([1.0, 2.1][0,1]);
    printf(myfunc()[0,2]);
    }
```

8.4.10 test-cov-matrix.yamml

```
matrix covmatrix (matrix m){
    matrix transposedm = trans(m);
    matrix oldm = m[:,:];
    int r;
    int c;
    for (r = 0; r < height(m); r = r + 1) {
        for (c = 0; c < width(m); c = c + 1) {
            float avg = mean(oldm[:, c:c]);
            m[r,c] = (m[r,c] - avg) / 2.0;
        }
    }
    return transposedm * m;
}

int main(){
matrix m = [ 10.0, 20.0, 30.0 ; 15.0, 30.0, 45.0 ];
printm(covmatrix(m)); //can't have hyphen in function name
}
```

8.4.11 test-filter2d.yamml

```
int main() {

    matrix M = [1.0, 2.0, 3.0 ;1.0, 2.0, 3.0; 5.0,6.0,10.0] ;
    matrix k = [1.0,1.0;1.0,1.0];
    matrix B = [1.0, 1.1, 1.2 ;1.3, 1.4, 1.5; 1.6,1.7,1.8];

    filter2D(M, B, -1, k);
    matrix result = B[1:height(B)-1,1:width(B)-1];
    printmat(result);

}
```

8.4.12 test-global_expr.yamml

```
float a = 5.0;
int b = 2 + 5 + 7;
```

```
str s = "hey world\n";
int main()
{
        float c = 7.8;
        printf(a + c);
        print(b);
}
```

8.4.13 test-globalvariablescope1.yamml

```
    float a = 5.0;
    int b = 2 + 5 + 7;
    str s = "hey world\n";
    int main()
    {
        float c = 7.8;
        printf(a + c);
        print(b);
    }
```

8.4.14 test-helloworld.yamml

```
int main() {
print(5);
}
```

8.4.15 test-helloworld2.yamml

```
int main() {
prints("helloworld");
}
```

8.4.16 test-if1.yamml

```
int main()
{
  if (true) print(42);
}
```

8.4.17 test-if2.yamml

```
int main()
{
  if (false) print(42);
  print(52);
}
```

8.4.18 test-if4.yamml

```
int main()
{
```

```
  /*test if local var works in if*/
  bool x = false;
  if (x==false) prints("got true, which is what I wanted");
  print(52);
}
```

8.4.19 test-inversetest.yamml

```
int main(){
    matrix A = [7.0,2.0,1.0;0.0,3.0,-1.0;-3.0,4.0,-2.0];
    matrix B = A[:,:];
    invert(A,B);
    printmat(B);

}
```

8.4.20 test-mateleassign1.yamml

```
int main() {
    matrix M = [1.0, 1.1; 51.1, 61.1; 1.0, 1.3];
    printm(M);
    prints("");
    M[0, 0] = 2.0;
    printm(M);
}
```

8.4.21 test-matmul1.yamml

```
int main()
{
    matrix M = [ 1.0, 2.0, 3.0, 4.0 ];
    matrix B = [1.0; 2.0; 3.0; 4.0];
    matrix C = M * B;
    printm(C);
}
```

8.4.22 test-matmul2.yamml

```
int main()
{
    matrix M = [ 1.0, 2.0; 3.0, 4.0 ];
    matrix B = [ 1.0, 2.0; 3.0, 4.0 ];
    printm(M);
    prints("");
    printm(B);
    prints("");
    matrix C = M * B;
    prints("");
    printm(C);
}
```

8.4.23 test-matrixaccess1.yamml

```
int main() {
    matrix M = [ 1.0, 1.1 ];
    printf(M[0, 1]);
}
```

8.4.24 test-matrixaccess2.yamml

```
int main() {
    printf([1.0, 2.1][0,1]);
}
```

8.4.25 test-matrixaccess3.yamml

```
matrix myfunc()
{
return [ 0.0, 2.0, 5.1 ];
}

int main()
{
printf(myfunc()[0,2]);
}
```

8.4.26 test-matrixassign1.yamml

```
int main()
{
matrix m = [ 2.0, 2.0, 3.0, 4.0 ];
}
```

8.4.27 test-matrixconsmul1.yamml

```
int main () {
    matrix M = [1.0,2.0,3.0,4.0];
    matrix B = M * 3.0;
    matrix C = 3.0 * M;
    printf(B[0,0]);
    printf(B[0,1]);
    printf(B[0,2]);
    printf(B[0,3]);
    printf(C[0,0]);
    printf(C[0,1]);
    printf(C[0,2]);
    printf(C[0,3]);

}
```

8.4.28 test-matrixdimensions.yamml

```
int main() {
```

```
    matrix M = [1.0, 1.1, 1.2; 1.1, 1.1, 1.2];
    print(width(M));
    print(height(M));
}
```

8.4.29 test-matrixelemult.yamml

```
int main () {
    matrix M = [1.0,2.0,3.0,4.0];
    matrix B = [1.0,2.0,3.0,4.0];
    matrix C = M .* B;
    printf(B[0,0]);
    printf(B[0,1]);
    printf(B[0,2]);
    printf(B[0,3]);
    printf(C[0,0]);
    printf(C[0,1]);
    printf(C[0,2]);
    printf(C[0,3]);
}
```

8.4.30 test-matrixempty.yamml

```
int main(){

    matrix M = empty(10,10);
    printmat(M);

}
```

8.4.31 test-matrixheight.yamml

```
int main() {
    matrix M = [1.0; 1.1; 1.1; 1.1; 1.1];
    print(height(M));
}
```

8.4.32 test-matrixmean.yamml

```
int main() {
    matrix M = [1.3, 1.0, 4.0];
    printf(mean(M));

}
```

8.4.33 test-matrixsum.yamml

```
int main() {
    matrix M = [1.4, 1.1];
    printf(sum(M));

}
```

8.4.34 test-matrixtrans.yamml

```
int main() {

    matrix M = [1.0, 1.1, 1.2 ;1.3, 1.4, 1.5] ;

    printf(M[0,1]);

}
```

8.4.35 test-matrixtrans2.yamml

```
int main() {

    matrix M = [1.0, 1.1, 1.2];
    printf(M[0,2]);
    print(width(M));
    print(height(M));

    matrix B = trans(M);

    printf(B[2,0]);
    print(width(B));
    print(height(B));

    // printf(M[3,3]);
    printf(B[1,0]);
    printf(M[0,1]);

    if (B[1, 0] == M[0, 1]){
        prints("True");
    }
    else{
        prints("False");
    }
}
```

8.4.36 test-matrixtrans3.yamml

```
int main() {

    matrix M = [1.0;1.2;1.3;1.4] ;
    printf(M[1,0]);
    matrix B = trans(M);
    print(width(M));
    print(height(M));

    print(width(B));
    print(height(B));
    printf(B[0,1]);

}
```

8.4.37 test-matrixwidth.yamml

```
int main() {
    matrix M = [1.0, 1.1, 1.1, 1.1];
    print(width(M));
}
```

8.4.38 test-mixedlocals1.yamml

```
int main()
{
int x = 5;
x = x + 1;
int y;
y = 6;
int z = x + y;
print(z);
}
```

8.4.39 test-mixedlocalsscoped1.yamml

```
int main()
{
int z;
int x = 5;
{
x = x + 1;
int y;
y = 6;
z = x + y;
}
print(z);
}
```

8.4.40 test-neg1.yamml

```
int main()
{
  print(-1);
}
```

8.4.41 test-printb.yamml

```
int main()
{
    float f1 = 0.1;
    float f2 = 0.2;
    float f3 = 0.1;
    float f4 = 0.101;
    float epsilon = 0.01;
    printb(true);
    printb(false);
    printb(1 == 1);
```

```
    printf(f1 - f2);
    printf(-epsilon);
    printb(((f1 - f2) > -epsilon) && ((f1 - f2) < epsilon));
    printb(((f3 - f4) > -epsilon) && ((f3 - f4) < epsilon));
    printb(true);
}
```

8.4.42 test-printglobalstr.yamml

```
str s = "hey world\n";
int main()
{
prints(s);

}
```

8.4.43 test-printmat.yamml

```
int main() {

    matrix M = [1.0, 1.1, 1.2; 1.3, 1.4, 1.5] ;
    trans(M);
    printm(M);
}
```

8.4.44 test-return1.yamml

```
int main()
{
    return 42;
}
```

8.4.45 test-scope1.yamml

```
    int main()
    {
    int z;
    int x = 5;
    {
        x = x + 1;
        int y;
        y = 6;
        z = x + y;
    }
print(z);
    }
```

8.4.46 test-slcomments.yamml

```
int main() {
    //printf(5.5);
    print(3);
```

```
    prints("hello");
}
```

8.4.47 test-stdlib1.yamml

```
    int main() {
        matrix M = [1.0, 1.1, 1.2; 1.1, 1.1, 1.2];
        print(width(M));
        print(height(M));
        matrix M = [1.3, 1.0, 4.0];
        printf(mean(M));
        matrix M = [1.4, 1.1];
        printf(sum(M));
        matrix M = [1.0, 1.1, 1.2];
        printf(M[0,2]);
        print(width(M));
        print(height(M));

        matrix B = trans(M);

        printf(B[2,0]);
        print(width(B));
        print(height(B));
        printf(B[1,0]);
        printf(M[0,1]);

        if (B[1, 0] == M[0, 1]){
            prints("True");
        }
        else{
            prints("False");
        }
}
```

8.4.48 test-udfunc1.yamml

```
int greater_than_x(int l, int x) {
    if (l > x){
        return l;
    }
    else {
        while (l <= x){
            print (l);
            l = l + 1;
        }
        return l;
    }
}

int main(){
    print(greater_than_x(3, 5));
}
```

8.4.49 yammlc-cv.sh

```
    #!/bin/bash

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C++ compiler
CXX="c++"

# Path to the yamml compiler.  Usually "./yamml.native"
# Try "_build/yamml.native" if ocamlbuild was unable to create a symbolic link.
YAMML="./yamml.native"
#YAMML="_build/yamml.native"

basename="$(basename -s .yamml "$1")"
"$YAMML" "$1" > "${basename}.ll" &&
"$LLC" "${basename}.ll" > "${basename}.s" &&
# "$CC" "-c" "io.cpp" &&
# "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o"
# Run "$CC" "-c" "io.cpp" &&
# Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o" &&
# clang++ "${basename}.s" io.cpp -o "${basename}.exe"
# rm -f io.o
"$CXX" "-c" "io.cpp" &&
"$CXX" -std=c++11 -I /usr/local/opt/opencv/include "-c" "opencv.cpp" &&
"$CXX" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o" "opencv.o" -l opencv_core -l opencv_i
rm -f io.o opencv.o
```

8.4.50 yammlc.sh

```
    #!/bin/bash

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C++ compiler
CXX="clang++"

# Path to the yamml compiler.  Usually "./yamml.native"
# Try "_build/yamml.native" if ocamlbuild was unable to create a symbolic link.
YAMML="./yamml.native"
#YAMML="_build/yamml.native"

basename="$(basename -s .yamml "$1")"
"$YAMML" "$1" > "${basename}.ll" &&
```

```
"$LLC" "${basename}.ll" > "${basename}.s" &&
# "$CC" "-c" "io.cpp" &&
# "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o"
# Run "$CC" "-c" "io.cpp" &&
# Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o" &&
# clang++ "${basename}.s" io.cpp -o "${basename}.exe"
# rm -f io.o
"$CXX" "-c" "io.cpp" &&
"$CXX" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o" &&
rm -f io.o
```

8.4.51 yamml.ml

```
    (* Top-level of the YAMML compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

module StringMap = Map.Make(String)

type action = Ast | Sast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./yamml.native [-a|-l|-c] [file.yamml]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf channel = Lexing.from_channel !channel in
  let rec ast channel =
    let (imports, g, f) = Parser.program Scanner.token (lexbuf channel) in
    let import (gl, fu) im =
      let ichannel = ref (open_in (match im with Ast.Import f -> f)) in
      let iast = ast ichannel in
      (fst iast @ gl, snd iast @ fu)
    in
    List.fold_left import (g, f) imports
  in
  match !action with
    Ast -> print_string (Ast.string_of_program (ast channel))
  | _ -> let sast = Semant.check (ast channel) in
    match !action with
      Ast     -> ()
    (* | Ast     -> print_string (Ast.string_of_program ast) *)
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
```

```
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)

(*
  Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
  *)
```

8.4.52 testall.sh

```
#!/bin/bash

# Regression testing script for YAMML
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the yamml compiler. Usually "./yamml.native"
# Try "_build/yamml.native" if ocamlbuild was unable to create a symbolic link.
YAMML="./yamml.native"
#YAMML="_build/yamml.native"

# 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] [.yamml files]"
    echo "-k   Keep intermediate files"
    echo "-h   Print this help"
    exit 1
}

SignalError() {
```

```
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo " $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    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 $*
    STATUS="$?"
    test "$?" -eq 0 || {
        SignalError "$1 failed on $*"
        return "$STATUS"
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.yamml//'`
    reffile=`echo $1 | sed 's/.yamml$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe ${basename}.out"
        &&
    Run "$YAMML" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
```

```
    # Run "$CC" "-c" "io.cpp" &&
    # Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "io.o" &&
    clang++ `pkg-config --cflags opencv` `pkg-config --libs opencv` "${basename}.s" io.cpp -o "${
        basename}.exe" &&

    Run "./${basename}.exe" > "${basename}.out" &&
    STATUS="$?" &&
    if [ -f "${basename}.status" ]; then
       test "$STATUS" -eq "$(cat "${basename}.status")"
    fi &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                           s/.yamml//'`
    reffile=`echo $1 | sed 's/.yamml$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$YAMML" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

while getopts kdpsh c; do
```

```
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
        h) # Help
            Usage
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.yamml tests/fail-*.yamml"
fi

for file in $files
do
    case $file in
        *test-*)
            Check $file 2>> $globallog
            ;;
        *fail-*)
            CheckFail $file 2>> $globallog
            ;;
        *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
done

exit $globalerror
\end{lstlistng}

\subsubsection{opencv.cpp}
\begin{verbatim}
    #include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/imgcodecs.hpp>
```

```
using namespace cv;

typedef struct {
	double *arr;
	int32_t r;
	int32_t c;
} matrix_t;

#define MAT(mat) \
	Mat(mat->r, mat->c, CV_64F, mat->arr, sizeof(double) * mat->c)

static inline matrix_t YMAT_1B(Mat mat)
{
	double *arr = (double *)malloc((mat.rows * mat.cols) * sizeof(double));
	matrix_t ymat = { .arr = arr, .r = mat.rows, .c = mat.cols };
	int i;

	assert(ymat.arr != NULL);
	for (i = 0; i < ymat.r * ymat.c; ++i)
		ymat.arr[i] = (double)((unsigned char *)mat.data)[i];

	return ymat;
}

extern "C" void sfilter2D(matrix_t *src, matrix_t *dst, int ddepth, matrix_t *kernel)
{
	filter2D(MAT(src), MAT(dst), ddepth, MAT(kernel));
}

extern "C" void sinvert(matrix_t *src, matrix_t *dst)
{
	invert(MAT(src),MAT(dst),DECOMP_LU);
}

extern "C" void simread(const char *filename, matrix_t *dst)
{
	Mat mat = imread(filename, IMREAD_GRAYSCALE);
	*dst = YMAT_1B(mat);
}

extern "C" bool simwrite(const char *filename, matrix_t *img)
{
	return imwrite(filename, MAT(img));
}
```