Twister

Annalise Mariottini (aim2120) Arushi Gupta (ag3309)
Anand Sundaram (as5209) Chuan Tian (ct2698)

February 8, 2017

1 Language Description

Twister is a language designed for matrix mathematics with basic support for file input and output. It is an imperative programming language with first-class functions and a sparse set of built-in types, including a matrix type.

2 Intended Applications

Twister is intended to be used for image manipulation with efficient implementations of linear algebra operations such as convolution. Twister programs may be used to read in images, sharpen, de-noise, and write out images. Twister can be used by anyone from selfie lovers to image designers. It implements a simple to use pipe operation that allows users to chain together alterations made to images. It also provides built-in support for element-wise matrix operations.

3 Parts of the Language

3.1 Types

Primitive Types

Scalar primitive types

Type	Description	Initialization Example
int	an integer	`int a = 5`
float	a float	`float a = 5.2`

Other primitive types

Type	Description	Initialization Example
bool	a boolean	`bool x = True`
char	a character	`char x = 'a'`

Complex Types

Type	Description	Attributes	Initialization Example
List	a mutable linked list	length	`List l = {'a', 'b','c'}`
String	a List of chars	length	`String s = 'hello'`
Tup	an n-tuple of value	$[i]$	`Tup t = (0, 'a', 5.2)`
Struct	a collection of specified types	—	`Struct pt = {int x; int y;}`
Matrix	a series of scalar arrays	$[i]$, dim$[i]$, dims	`Matrix m = [0,0,0;0,0,0]`

Object Typing

The List and Matrix objects have associated value types that are defined either upon initialization or upon being declared with a specific type. This typing may also be specified for function arguments and return values (see section 3.5 Built-in Functions).

```
Matrix M = [1,1;1,1]; // automatically typed to int
List<float> N; // manually typed to float
```

Additional Matrix Initializations

Multidimensional

Dimension	Initialization
2	`[0,0;0,0]`
3	`[0,0;0,0];[0,0;0,0]`
4	`[[0,0;0,0];[0,0;0,0]];[[0,0;0,0];[0,0;0,0]]`

Function Initializer

In addition to initializing a Matrix with an array literal, one may also initialize with an n-tuple and an iterative function to initialize individual values based on position. For a matrix of n dimensions, the last n arguments of this initializer function will represent the $(x, y, \ldots n)$ position currently being initialized. This function must return either int or float. If the user does not have a initializer function specified, the matrix will initialize to zeros.

```
Matrix M = ((3,2), fill_val(1)); // [1,1,1;1,1,1]
```

3.2 Operators

Arithmetic Operators

Operator	Description
$-$	the subtraction operator
$+$	the addition operator
$*$	the multiplication operator
/	the division operator
%%	the modulo operator
.operator	element-wise matrix application

The arithmetic operators may be used on scalars as expected. The $*$ operator may be used on matrices to perform matrix multiplication. All arithmetic operators may be preceded with a '.' to perform element-wise operations of one matrix upon another matrix of identical dimensions, or a scalar upon a matrix.

```
Matrix a = [2,2;2,2];
Matrix b = [1,0;0,1];
int x = 3;

b .+ x; // returns [4,3;3,4]
a .* b; // returns [2,0;0,2]
a * b; // returns [2,2;2,2]
```

Functional Operators

Operator	Description	Example
|	function composition	`transpose(A) | convolution(this, kernel, 1)`

The functional operator pipes the return value of the left-hand-side function into the 'this' keyword, to be used in the arguments of the right-hand-side function.

Logical Operators

Operator	Description	Examples
==	equality comparison operator	`1 == 1 // True`
>	greater than operator	`1 > 2 // False`
<	less than operator	`1 < 2 // True`
and	takes two bools and returns their AND	`(1 > 2) and (2 == 2) // False`
or	takes two bools and returns their OR	`(1 > 2) or (2 == 1) // True`
not	returns the negation of a boolean	`not (1 == 1) // False`

Bitwise Operators

Operator	Description	Example
&&	bitwise and	`1 && 2 // 0`
||	bitwise or	`1 || 3 // 3`
*|	bitwise xor	`1 *| 3 // 2`
<<	left shift	`1 « 1 // 2`
>>	right shift	`2 » 1 // 1`
!	bitwise not	`!0 // -1`

3.3 Flow Control

For Loops

A 'for' loop is defined as an iteration over an ordered list with a named variable to represent the current element.

```
for(elem in range(0,3)) {
    print(elem);
} \\ output:
\\ 0
\\ 1
\\ 2
```

If-Else Statements

Users can check booleans and conditionals with if statements, with an optional else block to run if the condition is not satisfied.

```
int a = 5;
if (a == 5) {
    print('a was 5');
} else {
    print('a was not 5');
} \\ output: a was 5
```

3.4 Built-in functions

Function	Description
map	applies an element-wise function to a Matrix
reduce	folds a function across a Matrix or List
fread	takes a file name as a string and outputs a Matrix
fwrite	takes a Matrix and a file name and write the Matrix to the file
print	takes a String and a file and writes the string to the file
range	takes two arguments a and b, returns a List of ints $= \{a, a+1, ..., b-1\}$

All functions are first-class objects and support currying and argument/return type specification.

```
fun sum = (x: int, y: int) -> int {
    return x + y;
};

fun sum_matrix = (m: Matrix<int>) -> Matrix<int> {
    return reduce(sum, m);
};

Matrix a = [1,2;3,4];

sum_matrix(a) \\ returns 10
```

4 Example Program

```
fun square = (x: int) -> int {
    return x * x;
};

fun square_matrix = (m: Matrix<int>) -> Matrix<int> {
    return map(square, m);
};

fun transpose2D = (image: Matrix<int>) -> Matrix<int> {
    Tup newDims = (image.dims[1], image.dims[0]);
    fun swapVals = (x: int, y: int) -> int {
        return image[y][x];
    };
    return Matrix(newDims, swapVals);
};

fun fill = (apply: int, x: int, y: int) -> int {
    if (x %% 2 == 0 and y %% 2 == 0) {
        return apply;
    } else {
        return 0;
    }
};

Matrix a = fread('\Home\image.bmp');
Matrix b = Matrix((2,2), fill(2)); \\ [0,0;0,2]

a = transpose2D(a) | square_matrix(this) | this * b;

fwrite('\Home\image2.bmp', b);
```