

# Extend Language Final Report

Ishaan Kolluri  
isk2108  
Project Manager

Jared Samet  
jss2272  
Language Guru

Nigel Schuster  
ns3158  
System Architect

Kevin Ye  
ky2294  
Tester

December 20, 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Inspiration & Use Cases . . . . .	5
	Inspiration . . . . .	5
	Complex Calculations Across Many Inputs . . . . .	5
	Flexibility . . . . .	5
<b>2</b>	<b>Language Usage Tutorial</b>	<b>7</b>
2.1	Setup . . . . .	7
2.2	Compiling and Running Extend Code . . . . .	7
2.3	Writing Extend Code - The Basics . . . . .	7
	Adjusting to Extend's Declarative Nature . . . . .	8
	Functions . . . . .	8
	Data Types . . . . .	8
	Variables . . . . .	9
	Variables vs. Ranges - Similar, but not the same . . . . .	9
	Function Parameters - Using Dimensions . . . . .	10
	Enough theory. Show me a function that does something! . . . . .	10
	Range Slicing & Selection . . . . .	10
	How is this like a spreadsheet? . . . . .	11
	The Hash Operator . . . . .	11
	Cell Evaluation, Side Effects, and Precedence Expressions . . . . .	12
	Operators . . . . .	12
	Arithmetic Operators . . . . .	12
	Bitwise Operators . . . . .	13
	Boolean Operators . . . . .	13
	String Concatenation . . . . .	13
	The "Where am I?" operators . . . . .	13
	The size and typeof operators . . . . .	13
	Conditionals . . . . .	13
	If-Then-Else . . . . .	13
	The Switch Expression . . . . .	13
	Import Statements . . . . .	14
2.4	Illustrating the Benefits of Extend . . . . .	14
2.5	Standard Library Functions . . . . .	15
	Basic Functions . . . . .	15
	The toString() Function . . . . .	15
	The Print Function . . . . .	15
	Math Functions . . . . .	15
	File I/O . . . . .	16
	Additional Standard Library Functions . . . . .	16

	Flatten . . . . .	16
	Match . . . . .	16
	Binary Search . . . . .	16
	Statistics Functions . . . . .	16
	Matrix Multiplication . . . . .	16
	Concatenation . . . . .	16
	Repeat . . . . .	16
	Split & Split to Range . . . . .	16
	Parsing Strings . . . . .	16
	Reverse . . . . .	17
	Trim Functions . . . . .	17
	Plotting Bar Charts . . . . .	17
<b>3</b>	<b>Language Reference Manual</b>	<b>18</b>
3.1	Introduction to Extend . . . . .	18
3.2	Structure of an Extend Program . . . . .	18
	Import Statements . . . . .	18
	Function Definitions . . . . .	18
	Global Variables . . . . .	19
	External Library Declarations . . . . .	19
	<b>main</b> function . . . . .	19
	Scoping and Namespace . . . . .	19
3.3	Types and Literals . . . . .	20
	Primitive Data Types . . . . .	20
	Ranges . . . . .	20
	Range Literals . . . . .	20
3.4	Expressions . . . . .	21
	Arithmetic Operators . . . . .	22
	Boolean Operators . . . . .	22
	Conditional Expressions . . . . .	24
	Ternary Expressions . . . . .	24
	Switch Expressions . . . . .	24
	Additional Operators . . . . .	25
	Function Calls . . . . .	25
	Range Expressions . . . . .	25
	Slices . . . . .	25
	Selections . . . . .	26
	Corresponding Cell . . . . .	26
	Selection Examples . . . . .	26
	Precedence Expressions . . . . .	27
3.5	Functions . . . . .	27
	Format . . . . .	27
	Variable Declarations . . . . .	27
	Formula Assignment . . . . .	27
	Combined Variable Declaration and Formula Assignment . . . . .	28
	Formula Assignment Errors . . . . .	28
	Parameter Declarations . . . . .	28
	Application on Ranges . . . . .	29
	Lazy Evaluation and Circular References . . . . .	29
	External Libraries . . . . .	30
3.6	Standard Library Reference . . . . .	30
3.7	Example Program . . . . .	30

<b>4</b>	<b>Project Plan</b>	<b>31</b>
4.1	Meetings . . . . .	31
4.2	Development Workflow . . . . .	31
	Github & Travis CI . . . . .	31
4.3	Project Feature Completion Timeline . . . . .	32
	Style Guide . . . . .	32
4.4	Language Evolution . . . . .	33
	The Interpreter . . . . .	33
4.5	Team Member Responsibilities . . . . .	33
<b>5</b>	<b>Extend's Internal Architecture</b>	<b>34</b>
5.1	The Extend Compiler . . . . .	35
	The Scanner . . . . .	35
	The Parser and Abstract Syntax Tree . . . . .	35
	The Transformer . . . . .	35
	The Semantic Analyzer . . . . .	36
	The Code Generator . . . . .	36
	The Linker . . . . .	36
5.2	Extend Runtime . . . . .	37
<b>6</b>	<b>Testing</b>	<b>40</b>
6.1	Feature Integration & Testing . . . . .	40
6.2	Regression Test Suite . . . . .	40
	Integration with Travis CI . . . . .	41
<b>7</b>	<b>Example Source Programs</b>	<b>42</b>
7.1	maybeCircular . . . . .	42
7.2	True Shooting Percentage . . . . .	42
<b>8</b>	<b>Reflection</b>	<b>44</b>
8.1	Ishaan . . . . .	44
8.2	Jared . . . . .	44
8.3	Nigel . . . . .	44
8.4	Kevin . . . . .	45
<b>9</b>	<b>Extend Code Listing</b>	<b>46</b>
9.1	scanner.mll . . . . .	46
9.2	parser.mly . . . . .	48
9.3	ast.ml . . . . .	52
9.4	transform.ml . . . . .	58
9.5	semant.ml . . . . .	66
9.6	codeGenTypes.ml . . . . .	69
9.7	codegen.ml . . . . .	74
9.8	linker.ml . . . . .	98
9.9	main.ml . . . . .	99
9.10	lib.c . . . . .	100
9.11	runtime.c . . . . .	106
9.12	stdlib.xtnd . . . . .	121
<b>10</b>	<b>Tests and Output</b>	<b>130</b>
<b>11</b>	<b>Git Logs</b>	<b>159</b>



# 1. Introduction

Extend is a declarative programming language meant to support spreadsheet-like functionality. It contains features such as side-effect-free values, immutability, and automatic formula adjustments relative to rows and columns. Extend is compiled to the LLVM (Low Level Virtual Machine) intermediate representation, which in turn is reduced to machine assembly. Extend takes inspiration from software such as Microsoft Excel, which allows users to link several formulae on dependent groups of data together, but takes this technology a step further by allowing users to encapsulate such calculations as functions.

## 1.1 Inspiration & Use Cases

### Inspiration

The design goal of our language was to be "a spreadsheet you can compile". Extend was conceptualized to address the limitations that prevented the spreadsheet environment from evolving into a compiled, flexible programming language. To create this, there were three main things that needed to be changed about the way interactive spreadsheets work:

- The language needs reusable functions as opposed to having to copy & paste a block of cells.
- Cell ranges need to be created with dynamic runtime-determined dimensions.
- Cells need to be able to contain composite values in addition to single numbers or strings.

With these changes in mind, we attempted to keep the semantics as similar as possible to traditional spreadsheet programs; this meant implementing a dynamically typed language that is tolerant of potential errors in its input data. Extend degrades gracefully in the presence of potential data errors.

Spreadsheet applications cannot be 'run' on different sets of input data. Extend was conceptualized as a language to create standalone executables that can be repeatedly run on multiple files, thereby removing the need to manually enter inputs. In building this language, our mission was to bring the best of spreadsheets and computation into one product.

### Complex Calculations Across Many Inputs

Extend is spiritually closer in behavior to Microsoft Excel than traditional imperative programming languages. The order of computation is determined implicitly by the language rather than explicitly by the developer. In addition, in one line of code, a single formula can be assigned to all the cells in a variable. The feature acts similarly to Python's list comprehension, or OCaml's `List.map` functionality.

### Flexibility

Extend allows the dimensions of ranges to be determined dynamically at runtime, and handles most type errors by degrading gracefully instead of crashing the program. The standard library that Extend delivers

includes a subset of the functions that are built into conventional spreadsheet applications. As many of these as possible were implemented in Extend itself.

## 2. Language Usage Tutorial

This will cover the configuration of the user's environment and the usage of Extend's features.

### 2.1 Setup

The Extend compiler requires that the OCaml Language and LLVM be installed on the host machine. Development was done in a virtual machine running the 64-bit Ubuntu operating system. In order to quickly get Extend up and running, please use [this virtual machine](#), which has been provided as part of the course.

After booting up the virtual machine, clone the Extend git repository:

```
1 git clone https://github.com/ExtendLang/Extend.git
```

### 2.2 Compiling and Running Extend Code

To build the Extend compiler, the first steps are the following.

```
1 cd Extend/  
2 make
```

If this does not successfully build, run `eval 'opam config env'`, which should configure the environment to use OPAM packages. Alternatively, add this command to your bash profile.

After running `make`, you should see a `main.byte` file. To compile and run an Extend program, we have provided a shell script to simplify the process for the user:

```
1 ./compile.sh samples/helloworld.xtnd
```

This should produce an `out` file. Running `./out` should successfully execute the program.

### 2.3 Writing Extend Code - The Basics

As is tradition, here is "Hello World" in Extend. The following program, `helloworld.xtnd`, illustrates a basic usage of the Extend language.

```
1 main(args) {  
2     return print_endline("Hello, World!");  
3 }
```

Below is a short tour of the features of Extend. More detail can be found in the next chapter - the Language Reference Manual.



## Adjusting to Extend's Declarative Nature

The biggest difference between Extend and most traditional programming languages is that the concept of an imperative statement does not exist. An Extend function consists solely of variable declarations, formula assignments, and a return expression. When a function is called, its **return** expression is evaluated, along with the values of any variables that the return expression depends on. In a traditional imperative language, the order of operations is determined explicitly by the developer; in Extend, the order is determined implicitly by the desired result.

The following file compiles and prints successfully.

```
1  main(args) {
2      foo := "Hello World!";    // Combined var declaration and formula assignment
3      return print_endline(foo); // Return expression is a call to print_endline()
4  }
```

The next file compiles, but might surprise you by not printing anything.

```
1  main(args) {
2      foo := "Hello World!";    // Formula assigned, never evaluated
3      bar := print_endline(foo); // Formula assigned, never evaluated
4      return 0;                 // Return expression is just 0
5  }
```

And this file isn't a grammatical Extend program:

```
1  main(args) {
2      foo := "Hello World!"; // OK
3      print_endline(foo);    // Syntax error – not a declaration or assignment
4      return foo;
5  }
```

As illustrated, Extend only evaluates what is needed to produce the value required by **return**. Any non-essential declarations or formula assignments will not be evaluated by the program.

## Functions

An Extend program is mostly composed of functions, declared with the usual syntax **f(x, y, ...)**. Each Extend program must have a **main()** function taking one argument, as shown above in "Hello World". Inside the function, this parameter will contain the command-line arguments. A function is composed of variable declarations and formula assignments and concludes with the **return** statement. It can return a value of any of the types discussed below, and it doesn't always need to return the same type. Note that the **return** statement is always the last statement in the function.

## Data Types

Extend has three primitive data types: Number, String, and **empty**; and one composite type, Range. An example of each is shown below.

```
1  myNumber    := 5;
2  myString    := "Hello World";
3  myEmpty     := empty;
4  my2x3Range  := {3, 4, "five"; "a", "b", "c"};
```

## Variables

In Extend, **variables** are composed of cells to which formulas are assigned. The first time (and only the first time!) an individual cell is referenced by an expression, its value is calculated according to its assigned formula. A cell's value is not calculated if the cell is never referred to, and is never recalculated; all cell values are immutable. A cell's value can be any of Extend's types, and different cells of a single variable can have different types.

```
1      [1,2] foo; // Declares a variable with 1 row and 2 columns (2 cells total)
2      [1,3] bar := 4; // Declares a variable with 1 row and 3 columns and
3                      // assigns the literal value 4 as the formula for each cell
4      [1,2] baz;           // Declares a 1x2 variable baz
5      baz[0,0] = "first";   // Assigns literal "first" as the formula for the
6      baz[0,1] = 1 + 1;     // 1st cell and the expression 1+1 for the 2nd cell
7      life := 6, universe := 7; // Declares 1x1 variables life and universe
8      answer := life * universe; // Declares a 1x1 variable the_answer and assigns
9                      // the formula life * universe to its sole cell
10     [1,10] half_and_half; // Declares a 1x10 variable half_and_half
11     half_and_half[0,0:5] = "milk"; // Assigns "milk" to the first five cells
12     half_and_half[0,5:10] = "cream"; // and "cream" to the second five cells
```

**Note** that we declare a variable and assign a formula to all of its cells in a single line with `:=`. If the variable has already been declared, a formula must be assigned using `=` instead of `:=`. As illustrated in this example, a single formula can be assigned to multiple cells of a variable with the slice syntax. The converse is not true: multiple formulas applying to a single cell will cause a runtime error. The contents of the slice, as well as the dimensions of the variable, can be any expression that evaluates to a number, not just a literal number. For example, this code snippet assigns the dimensions based on the `howBig()` function and the "left" and "right" formulas based on the `breakpoint()` function:

```
1      breakpoint() { return 7; }
2
3      howBig() { return 11; }
4
5      foo_func() {
6          [1,howBig()] foo;
7          foo[0, :breakpoint()] = "left";
8          foo[0, breakpoint():-1] = "right";
9          foo[0, -1] = "last";
10         return foo;
11     }
```

This example also illustrates that the start (or end) index of a slice can be omitted if the developer wants the formula to apply from the beginning (or to the end) of the dimension, and that negative numbers can be used in a slice to count backwards from the end. The first time a variable is referred to (directly or indirectly) by the return expression, its dimensions and the formula assignment slices are computed; from that point on, they never change. A subtle point in the example above: the `howBig()` function is invoked once, but the `breakpoint()` function is actually called twice: once for the "left" formula, and once for the "right" formula.

### Variables vs. Ranges - Similar, but not the same

A variable is not a data type; it is a collection of one or more cells with assigned formulas. A range is a value, which is internally implemented as a pointer to a subset of a variable's cells. A range is always composed of more than one value; a variable may have a single cell. The variable "backing" a range may not have been explicitly defined by the developer; for example, range literals are implemented using an anonymous variable.

## Function Parameters - Using Dimensions

Function arguments can be signed with dimensions. You can use these in two different ways, depending on what your function is doing. As a convenient way to find out the size of a range argument, just give the dimensions names:

```
1     foo([m,n] arg){
2         return m * n; // m and n initialized through arg
3     }
```

You can hardcode dimensions; if your function is called with a range whose dimensions don't match, a runtime error will occur:

```
1     determinant([2,2] arg){
2         return arg[0,0] * arg[1,1] - arg[0,1] * arg[1,0];
3     }
```

You can also combine these two mechanisms, by repeating a variable name:

```
1     betterBeSameSize([m,n] arg1, [m,n] arg2) {
2         return "I guess they were the same size."; // Error if they were different
3     }
```

## Enough theory. Show me a function that does something!

This function adds its two arguments.

```
1     add(x, y) {
2         return x + y;
3     }
```

Come on, a real function.

```
1     euclideanDistance([1,2] ptA, [1,2] ptB) {
2         return sqrt((ptA[0] - ptB[0]) ** 2 + (ptA[1] - ptB[1]) ** 2);
3     }
```

Tell me about that bit where you wrote ptA[0]!

## Range Slicing & Selection

The euclideanDistance() function above used a selection to extract the individual values from a range. ptA[0] is the first value of ptA and ptA[1] is the second value. Although ranges have rows and columns, you only need to give one index if a range is a vector—Extend will figure out what you mean. You can also get a slice, with essentially the same syntax as Python:

```
1     addTheFirstThreeElements([1,n] some_vector) {
2         return sum(some_vector[:3]);
3     }
```

If you're dealing with a 2-D range, you can get a rectangle by slicing both the rows and the columns.

```
1     topLeftCorner(m) {
2         return m[:2,:2] // Returns a 2x2 range with m[0,0], m[0,1], m[1,0], m[1,1]
3     }
```

## How is this like a spreadsheet?

Here's the Extend equivalent of this spreadsheet:

	A	B	C	D	E
1		Revenue	Cost	Profit	
2	Q1	\$82,500	\$80,000	\$2,500 =B2-C2	
3	Q2	\$97,800	\$105,000	-\$7,200 =B3-C3	
4	Q3	\$560,000	\$130,000	\$430,000 =B4-C4	

```
1  calcProfit([n,1] revenue, [n,1] cost) {
2    [n,1] profit := revenue[[0]] - cost[[0]];
3    return profit;
4  }
5  main(args) {
6    revenue := {82500; 97800; 560000};
7    cost := {80000; 105000; 130000};
8    profit := calcProfit(revenue, cost);
9    return print_endline(profit);
10 }
```

Writing `revenue[[0]]` and `cost[[0]]` instead of `revenue[0]` and `cost[0]` means that the *n*th cell of profit is calculated by subtracting the *n*th cells of cost from the *n*th cell of revenue; the number inside the brackets gets added to the row index of the left-hand-side cell. Here's how to calculate the change in profits from one quarter to the next:

A	B	C	D
	Profit	Profit Growth	
Q1	\$2,500		
Q2	-\$7,200	-\$9,700 =B3-B2	
Q3	\$430,000	\$437,200 =B4-B3	

```
1  calcProfitGrowth([n,1] profits) {
2    [n,1] profitGrowth := profits[[0]] - profits[[-1]];
3    return profitGrowth;
4  }
5  main(args) {
6    profits := {2500;-7200;430000};
7    return print_endline(calcProfitGrowth(profits));
8  }
```

Don't worry about the first cell - it'll be **empty**, not a program-ending `ArrayIndexOutOfBoundsException`. The selection syntax is very flexible; you can mix and match absolute and relative indexes and slices and omit the ones you don't need. There's a lot more examples in the language reference manual, but hopefully that should get you started! There's just one more special way you should know about to make a selection, since it's probably the most common selection you'll need.

## The Hash Operator

The hash operator gets the cell that's in "the equivalent place" of the cell whose formula is being calculated. Here's the quick way to add two matrices:

```
1  matrixAdd([m,n] arg1, [m,n] arg2) {
2    [m,n] result := #arg1 + #arg2;
3    return result;
4  }
```

And here's one more example to show its flexibility, with the spreadsheet equivalent:

	A	B	C	D
1		1	2	3
2	10	11	12	13
3	20	21	22	23
4	30	31	32	33

```
1 hashAdd([1,n] arg1, [m,1] arg2) {  
2   [m,n] result := #arg1 + #arg2;  
3   return result;  
4 }
```

If you call `hashAdd` with `{1,2,3}` as the first argument and `{10;20;30}` as the second argument, your result will be the matrix in the image. Enjoy making selections!

## Cell Evaluation, Side Effects, and Precedence Expressions

It's time for a little more theory. As mentioned before, a cell's value is calculated at most once. It is evaluated when it is the only cell selected from a variable, or when a selection containing the cell is assigned as a range to another cell. In general, the language is designed so you don't have to think about this! However, if a cell formula calls a function with side effects, it's important to keep in mind that it will only be evaluated once for each cell with that formula.

Another feature related to side effects is the precedence expression. If you want to call a function such as `print_endline()` for its side effects, but don't want it to be your return statement, you can use a precedence expression (written with the `->` operator) to force the evaluation of one expression before another. For example, to display a prompt before asking the user for input, you could write:

```
1 speed := print_endline("What is the air-speed velocity of an unladen swallow?")  
2       -> readline(STDIN);
```

A precedence expression calculates the first expression, discards the result, and evaluates to the second expression. Putting it all together, the following example should help clarify how cell evaluation is performed:

```
1 main(args) {  
2   foo := print_endline("Once") -> 2;  
3   bar := foo + foo;  
4   return print_endline(bar);  
5 }
```

This program prints "Once" and then prints 4. Before calling `print_endline`, Extend calculates the value of `bar`, which in turn requires the value of `foo` (twice). The first time `foo`'s value is calculated, `print_endline()` is called with the argument "Once", and then `foo` evaluates to the constant 2. The second time that `foo`'s value is required to calculate `bar`, it's already available: it is 2. Therefore, `print_endline("Once")` is not called a second time.

## Operators

Extend includes a comprehensive set of operators. Each category is listed in order of precedence. A more detailed explanation of each operator can be found in the Language Reference Manual.

### Arithmetic Operators

- Unary Operations: -
- Binary Operations: \*\*, \*, /, %, +, -

## Bitwise Operators

- Unary Operations: `~`
- Binary Operations: `<<`, `>>`, `&`, `|`, `^`

## Boolean Operators

- Unary Operations: `!`
- Binary Operations: `==`, `!=`, `<`, `>`, `<=`, `>=`, `&&`, `||`

## String Concatenation

Note that the `+` symbol can be used to perform concatenation between two strings.

```
1 "Hello " + "World\n"
```

## The "Where am I?" operators

Extend has the `row()` and `column()` functions, which respectively return the row and column of the left-hand-side cell whose formula is being calculated.

## The size and typeof operators

Extend offers a `typeof(expr)` operator, which takes an expression and returns Number, String, Range, or Empty (as a string). It also has the `size(expr)` operator, which returns the dimensions of its argument as a 1 x 2 range.

## Conditionals

There are two types of conditional expressions: the if-then-else (ternary) conditional and a `switch` expression.

### If-Then-Else

The two equivalent ways to write the ternary expression are as follows:

C/Java style: `condition ? expr_if_true : expr_if_false`  
Spreadsheet style: `if(condition, expr_if_true, expr_if_false)`

The predicate is always evaluated; only one of `expr_if_true` or `expr_if_false` will be evaluated—or neither, if the predicate is `empty`.

### The Switch Expression

Below is an example of the switch expression used in a function:

```
1 odd_or_even(foo) {  
2   return switch(foo % 2) {  
3     case 0: "Even";  
4     case 1: "Odd";  
5     default: "Not an integer";  
6   };  
7 }
```

In the example above, the `switch` expression used `foo % 2` as an argument; however, this is not required, so a `switch` expression can be used (as in Go) as a replacement for a sequence of if-then-else conditionals.

## Import Statements

In Extend, you can import other Extend files at the top of your program via relative directory path. The use case is below:

```
1 import "../programs/stat_library.xtnd"
```

## 2.4 Illustrating the Benefits of Extend

Excel and Google Sheets are pretty easy to use. Why go to all this trouble? Spreadsheet applications require the use of manual input in order to apply the same calculation to a different set of data. Extend aims to tackle this problem by offering portability. Below is an example of a spreadsheet user calculating the unit vector of a column vector:

	A	B	C	D	E
1	1	1			0.050965
2	2	4			0.101929
3	3	9			0.152894
4	4	16			0.203859
5	5	25			0.254824
6	6	36			0.305788
7	7	49			0.356753
8	8	64			0.407718
9	9	81			0.458682
10	10	100			0.509647
11		=A!*A!	385	19.62142	=A!/\$D\$11
12			=SUM(B1:B10)	=C11^0.5	

The Excel user must manually input the data, and additionally make space for the intermediate steps of the calculation. If the number of elements of the vector were changed, the formulas would need to be changed in the spreadsheet; similarly, if you needed to do this on a second vector, you would have to copy and paste the cells doing intermediate calculations. Below is the equivalent function in Extend, written to work on any column vector that is passed in:

```
1 normalize_column_vector([m,1] arg) {
2   [m,1] squared_lengths := #arg * #arg, normalized := #arg / vector_norm;
3   vector_norm := sqrt(sum(squared_lengths));
4   return normalized;
5 }
```

Another simple example is concatenating a row of strings of variable length with a common delimiter. This in an entirely manual operation for the spreadsheet user; a step-by-step attempt is shown below.

	A	B	C	D	E	F
1	hello	world	hello again	,	<- comma, space	
2						
3	hello,	<- This fails.				
4	=CONCATENATE(A1:C1, D1)					
5						
6	hello	hello, world	hello, world, hello again			
7	=A1	=CONCATENATE(A1,D1,B1)	=CONCATENATE(B6,D1,C1)			

Performing a delimiter 'join' like the above can be performed in a simple program in Extend without knowing the size of the row. The following function, which is included in the Extend standard library, performs this on arguments of any size and can be reused throughout the program.

```
1  main(args) {
2      bar := {"Hello", "Goodbye", "Hello Again"};
3      str := ", ";
4      return print_endline(concatRow(bar, str)); // prints "Hello, Goodbye, Hello Again"
5  }
6
7  concatRow([1,n] cells, joiner) {
8      [1,n] accum;
9      accum[0,0] = #cells;
10     accum[0,1:] = accum[[-1]] + joiner + #cells;
11     return accum[-1];
12 }
```

As evidenced above by simple examples, Extend offers flexibility that is significantly harder to achieve with conventional spreadsheet applications. As the nature of the data grows in complexity and variety, Extend's value increases.

## 2.5 Standard Library Functions

Extend offers an assortment of standard library functions. The standard library is automatically imported into each Extend program.

A complete listing of the functions in the standard library can be found in the Language Reference Manual; some of the more popular ones are listed below.

### Basic Functions

#### The toString() Function

The `toString()` function takes an argument and renders its value as a string.

```
1  return "Hello " + toString(14); // "Hello 14"
```

#### The Print Function

As used throughout this tutorial, the `print_endline` function is used to print an expression with a newline.

### Math Functions

Borrowing from C's standard library math functions, Extend offers: `sin`, `cos`, `tan`, `acos`, `asin`, `atan`, `sinh`, `cosh`, `tanh`, `exp`, `log`, `log10`, `sqrt`, `ceil`, `fabs` and `floor`.

```
1  main(args) {
2      bar := sqrt(16);
3      return print_endline(bar) -> 0; // Prints 4 to stdout
4  }
```



## File I/O

Extend has `open`, `close`, `read`, and `write` functions to interact with files. Usage is as follows:

```
1   main(args) {
2       return write(STDOUT, read(open("test_file.txt", "r"),5)) -> 0; // Writes 5
        characters from test_file.txt to stdout
3   }
```

## Additional Standard Library Functions

### Flatten

The `flatten` function turns a rectangular range into a long row vector.

```
1   flatten({1,2,3; 4,5,6}) // yields {1,2,3,4,5,6}
```

### Match

The `match` function takes a row or column vector and a value, and locates the index of that value, if applicable

### Binary Search

The `bsearch` function will search a sorted column vector for a value.

### Statistics Functions

Extend additionally offers basic statistical functions such as `sum`, `max`, `avg`, and `stddev`.

### Matrix Multiplication

The `mmult` function multiplies two compatible rectangular ranges together in matrix-fashion.

### Concatenation

The `concatRow` function takes a column vector and a delimiter and returns a string of each element in the vector joined by the delimiter.

### Repeat

The `repeat` function takes a string and number `x`, and returns a string where the argument string is repeated `x` times.

```
1   repeat("Hello", 3) // "HelloHelloHello"
```

### Split & Split to Range

The `split` function takes a string and a splitter and returns a vector of the delimited characters. Expanding on this, the `splittoRange` function takes a string, row splitter, and column splitter and returns a rectangular range with the characters delimited by the splitters.

### Parsing Strings

The `parseString` function leverages the above two functions to create an actual range with the characters parsed as numeric values.

## Reverse

Reverse takes a string and reverses it.

## Trim Functions

The `trim` function removes preceding and following whitespace from a string and returns the new string. Similarly, the `ltrim` function removes preceding whitespace, and `rtrim` the following whitespace.

## Plotting Bar Charts

Providing a file handle, a row vector, and an equivalently sized vector of labels to `bar_chart` will allow the user to write a bar graph in GIF form to the file descriptor.

## 3. Language Reference Manual

### 3.1 Introduction to Extend

Extend is a domain-specific programming language used to designate ranges of cells as reusable functions. It is a dynamically-typed, statically-scoped, declarative language that uses lazy evaluation to carry out computations. Once computed, all values are immutable. In order to offer the best performance, Extend compiles down to LLVM.

Extend's syntax is meant to provide clear punctuation and easily understandable cell range access specifications, while borrowing elements from languages with C-style syntax for ease of development. Despite these syntactic similarities, the semantics of an Extend program have more in common with a spreadsheet such as Microsoft Excel than imperative languages such as C, Java or Python.

### 3.2 Structure of an Extend Program

An Extend program consists of one or more source files. A source file can contain any number of import directives, function definitions, global variable declarations, and external library declarations, in any order.

#### Import Statements

Import statements in Extend are written with `import`, followed by the name of a file in double quotes, and terminated with a semicolon. The syntax is as follows:

```
1 import "string.xtnd";
```

Extend imports act like `#include` in C, except that multiple imports of the same file are ignored. The imports are all aggregated into a single namespace.

#### Function Definitions

Function definitions comprise the bulk of an Extend program. In short, a function consists of a set of variable declarations, formula assignments, and a return expression. Each variable consists of cells; the values of each cell are, if necessary, calculated according to formulas which each apply to a specified subset of the cells. Each cell value, once calculated, is immutable. A couple examples follow for context; functions are described in detail in section 3.5.

```
1 isNumber(x) {
2     return typeof(x) == "Number";
3 }
4
5 sum_column([m,1] rng) {
6     /* Returns the sum of the values in the column, skipping any values that are non-
7        numeric */
7     [m,1] running_sum;
8     running_sum[0,0] = #rng;
```

```

9   running_sum[1:,0] = running_sum[[-1],] + (isNumber(#rng) ? #rng : 0);
10  return running_sum[-1];
11 }

```

## Global Variables

In essence, global variable declarations function as constants in Extend. They are written with the keyword **global**, followed by a variable declaration in the combined variable declaration and assignment format described in section 3.5. As with local variables, the cell values of a global variable, once computed, are immutable. A few examples follow:

```

1  global pi := 3.14159265359;
2  global num_points := 24;
3  global [num_points,1]
4    circle_x_vals := cos(2 * pi * row() / num_points),
5    circle_y_vals := sin(2 * pi * row() / num_points);

```

## External Library Declarations

An external library is declared with the **extern** keyword, followed by the name of an object file in double quotes, followed by a semicolon-delimited list of external function declarations enclosed by curly braces. A library declaration informs the compiler of the functions' names and signatures and instructs the compiler to link the object file when producing an executable. An external function declared as **foo** will call an appropriately written C function **extend\_foo**. An example follows:

```

1  extern "mylib.o" {
2    foo(arg1, arg2);
3    bar();
4  }

```

This declaration would cause the compiler to link **mylib.o** and would make the C functions **extend\_foo** and **extend\_bar** available to Extend programs as **foo** and **bar** respectively. The required signature and format of the external functions is specified precisely in section 3.5.

## main function

When a compiled Extend program is executed, the **main** function is evaluated. All computations necessary to calculate the return value of the function are performed, after which the program terminates. The **main** function must be a function of a single argument, conventionally denoted **args**, which is guaranteed to be a 1-by-n range containing the command line arguments.

## Scoping and Namespace

For functions and for global variables, there is a single namespace that is shared between all files composing an Extend program, and they are visible throughout the entire program. Functions declared in external libraries share this namespace as well. For a local variable, the scope is the entire body of the function in which it is defined. Functions may declare local variables sharing a name with a global variable; inside that function, the name will refer to the local variable.

```

1  global x := "I'm a global";
2
3  foo() {
4    y := x; // Scope of x is entire function
5    x := "In here I'm a local";
6    return y; // Returns "In here I'm a local"

```

```

7 }
8
9 bar(x) {
10     return x; // Parameters mask globals; returns argument
11 }
12
13 baz() {
14     return x; // Returns "I'm a global"
15 }

```

### 3.3 Types and Literals

Extend has three primitive data types, **Number**, **String**, and **Empty**, and one composite type, **Range**.

#### Primitive Data Types

A **Number** is an immutable primitive value corresponding to a double-precision 64-bit binary format IEEE 754 value. Numbers can be written in an Extend source file as either integer or floating point constants; both are represented internally as floating-point values. There is no separate type representing an integer.

A **String** is a immutable primitive value that is internally represented a C-style null-terminated byte array corresponding to ASCII values. A String can be written in an Extend source file as a sequence of characters enclosed in double quotes, with the usual escaping conventions. Extend does not allow for slicing of strings to access specific characters; access to the contents of a string will only be available through standard library functions.

The **Empty** type can be written as the keyword **empty**, and serves a similar function to NULL in SQL; it represents the absence of a value.

Primitive Data Types	Examples
Number	42 or -5 or 2.71828 or 314159e-5
String	"Hello, World!\n" or "foo" or ""
Empty	empty

#### Ranges

Extend has one composite type, **Range**. A range borrows conceptually from spreadsheets; it is a group of cells with two dimensions, described as rows and columns. Each cell is assigned a formula that either evaluates to a Number, a String, **empty**, or another Range. Cell formulas are described in detail in section 3.5. A range can either be declared as described in section 3.5 or with a range literal expression. Ranges can be nested arbitrarily deeply and can be used to represent (immutable) lists, matrices, or more complicated data structures.

#### Range Literals

A range literal is a semicolon-delimited list of rows, enclosed in curly brackets. Each row is a comma-delimited list of numbers, strings, or range literals. A few examples follow:

```

1 legal_ranges() {
2     r1 := {"Don't"; "Panic"}; // two rows, one column
3     r2 := {"Don't", "Think", "Twice"}; // one row, three columns
4     r3 := {1,2,3;4,5,6;7,8,9}; // three rows, three columns
5     r4 := {"Hello";0,1,2,3,4}; // two rows, five columns

```

```

6   r5 := {{{{{1}}}}}; // one row, one column
7   r7 := {-1.5,-2.5,{-2,"nested"},-3.5}; // one row, four columns
8   return
9       print_endline(r1) ->print_endline(r2) ->print_endline(r3) ->
10      print_endline(r4) -> print_endline(r5) -> print_endline(r7);
11 }
12
13 main(args) {
14     return legal_ranges();
15 }

```

## 3.4 Expressions

Expressions in Extend allow for arithmetic and boolean operations, function calls, conditional branching, extraction of contents of other variables, string concatenation, and determination of the location of the cell containing the expression. The sections for boolean and conditional operators refer to *truthy* and *falsey* values: the **Number** 0 is the only falsey value; all other values are *truthy*. As **empty** represents the absence of a value, it is neither *truthy* nor *falsey*.

## Arithmetic Operators

The arithmetic operators listed below take one or two expressions and return a number, if both expressions are Numbers, or **empty** otherwise. Operators grouped within the same inner box have the same level of precedence, and are listed from highest precedence to lowest precedence. All of the binary operators are infix operators, and, with the exception of exponentiation, are left-associative. Exponentiation, bitwise negation, and unary negation are right-associative. All of the unary operators are prefix operators. The bitwise operators round their operands to the nearest signed 32-bit integer (rounding half to even) before performing the operation and evaluate to a Number.

Operator	Description	Definition
<code>~</code>	Bitwise NOT	Performs a bitwise negation on the binary representation of an expression.
<code>-</code>	Unary negation	A simple negative sign to negate expressions.
<code>**</code>	Power	Returns the first expression raised to the power of the second expression
<code>*</code>	Multiplication	Multiplies two expressions
<code>/</code>	Division	Divides first expression by second.
<code>%</code>	Modulo	Finds the remainder by dividing the expression on the left side of the modulo by the right side expression.
<code>&lt;&lt;</code>	Left Shift	Performs a bitwise left shift on the binary representation of an expression.
<code>&gt;&gt;</code>	Right Shift	Performs a sign-propagating bitwise right shift on the binary representation of an expression.
<code>&amp;</code>	Bitwise AND	Performs a bitwise AND between two expressions.
<code>+</code>	Addition	Adds two expressions together.
<code>-</code>	Subtraction	Subtracts second expression from first.
<code> </code>	Bitwise OR	Performs a bitwise OR between two expressions.
<code>^</code>	Bitwise XOR	Performs a bitwise exclusive OR between two expressions.

```
1 easy() {
2   return 3 - -3 ** 2 %5; //-1
3 }
4 g_eazy() {
5   return (((1 << 2 | 1) << 2) | 1) << 1; //42
6 }
```

## Boolean Operators

These operators take one or two expressions and evaluate to **empty**, 0 or 1. Operators grouped within the same inner box have the same level of precedence and are listed from highest precedence to lowest precedence. All of these operators besides logical negation are infix, left-associative operators. The logical AND and OR operators feature short-circuit evaluation. Logical NOT is a prefix, right-associative operator. Besides logical NOT, all boolean operators have lower precedence than all arithmetic operators. For Strings, the boolean operators `<`, `<=`, `>`, and `>=` implement case-sensitive lexicographic comparison.

Operator	Description	Definition
!	Logical NOT	Evaluates to 0 or 1 given a truthy or falsey value respectively. <code>!empty</code> evaluates to <code>empty</code> . It has equal precedence with <code>and</code> and unary minus.
==	Equals	Always evaluates to 0 if the two expressions have different types. If both expressions are primitive values, evaluates to 1 if they have the same type and the same value, or 0 otherwise. If both expressions are ranges, evaluates to 1 if the two ranges have the same dimensions and each cell of the first expression == the corresponding cell of the second expression. <code>empty == empty</code> evaluates to 1. Strings are compared by value.
!=	Not equals	<code>x != y</code> is equivalent to <code>!(x == y)</code> .
<	Less than	If the expressions are both Numbers or both Strings and the first expression is less than the second, evaluates to 1. If the expressions are both Numbers or both Strings and the first expression is greater than or equal to the second, evaluates to 0. Otherwise, evaluates to <code>empty</code> .
>	Greater than	Equivalent rules about typing as for <.
<=	Less than or equal to	Equivalent rules about typing as for <.
>=	Greater than or equal to	Equivalent rules about typing as for <.
&&	Short-circuit Logical AND	If the first expression is falsey or <code>empty</code> , evaluates to 0 or <code>empty</code> respectively. Otherwise, if the second expression is truthy, falsey, or <code>empty</code> , evaluates to 1, 0, or <code>empty</code> respectively.
	Short-circuit Logical OR	If the first expression is truthy or <code>empty</code> , evaluates to 1 or <code>empty</code> respectively. Otherwise, if the second expression is truthy, falsey, or <code>empty</code> , evaluates to 1, 0, or <code>empty</code> respectively.

```

1 somethings_false() {
2   return !1 != !1 || 4 <= 3;
3 }
4 somethings_empty() {
5   return empty || empty <= !3 || 5 > 3;
6 }
7 somethings_true() {
8   return 6 > 2 && !(1 == !1);
9 }

```



## Conditional Expressions

There are two types of conditional expressions: a simple ternary if-then-else expression and a **switch** expression which can represent more complex logic.

### Ternary Expressions

A ternary expression, written either as `cond-expr ? expr-if-true : expr-if-false` or, equivalently, `if(cond-expr, expr-if-true, expr-if-false)` evaluates to `expr-if-true` if `cond-expr` is truthy, or `expr-if-false` if `cond-expr` is falsey. If `cond-expr` is empty, the expression evaluates to `empty`. Both `expr-if-true` and `expr-if-false` are mandatory. `expr-if-true` is only evaluated if `cond-expr` is truthy, and `expr-if-false` is only evaluated if `cond-expr` is falsey. If `cond-expr` is empty, neither expression is evaluated. The ternary operator `? :` has the lowest precedence level of all operators.

### Switch Expressions

A **switch** expression takes a optional condition, and a list of cases and expressions that the overall expression should evaluate to if the case applies. In the event that multiple cases are true, the expression of the first matching case encountered will be evaluated. An example is provided below:

```
1 switch_example(foo) {
2   return switch (foo) {
3     case 2: "foo is 2";
4     case 3,4: "foo is 3 or 4";
5     default: "none of the above";
6   };
7 }
8
9 alternate_format(foo) {
10  return switch {
11    case foo == 2:
12      "foo is 2";
13    case foo == 3, foo == 4:
14      "foo is 3 or 4";
15    default:
16      "none of the above";
17  };
18 }
```

The format for a **switch** statement is the keyword **switch**, optionally followed by pair of parentheses containing an expression **switch-expr**, followed by a list of case clauses enclosed in curly braces and delimited by semicolons. A case clause consists of the keyword **case** followed by a comma-separated list of expressions **case-expr1** [, **case-expr2**, [...]], a colon, and an expression **match-expr**, or the keyword **default**, a colon, and an expression **default-expr**. If **switch-expr** is omitted, the **switch** expression evaluates to the **match-expr** for the first case where one of the **case-exprs** is truthy, or **default-expr** if none of the **case-exprs** apply. If **switch-expr** is present, the **switch** expression evaluates to the **match-expr** for the first case where one of the **case-exprs** is equal (with equality defined as for the `==` operator) to **switch-expr**, or **default-expr** if none of the **case-exprs** apply.

The **switch** expression can be used to compactly represent what in most imperative languages would require a long string such as `if (cond1) {...} else if (cond2) {...}`. The **switch** operator is internally converted to an equivalent (possibly nested) ternary expression; as a result, it features short-circuit evaluation throughout.

## Additional Operators

There are four additional operators available to determine the size and type of other expressions. In addition, the infix `+` operator is overloaded to perform string concatenation.

Operator	Description	Definition
<code>size(expr)</code>	Dimensions	Evaluates to a Range consisting of one row and two columns; the first cell contains the number of rows of <code>expr</code> and the second contains the number of columns. If <code>expr</code> is a Number, a String, or Empty, both cells will contain 1.
<code>typeof(expr)</code>	Value Type	Evaluates to "Number", "String", "Range", or "Empty".
<code>row()</code>	Row Location	No arguments; returns the row of the cell that is being calculated
<code>column()</code>	Column Location	No arguments; returns the column of the cell that is being calculated
<code>+</code>	String concatenation	"Hello, " + "World!\n" == "Hello, World!\n"

Given `[5,5]foo`, then `foo[1,4] = row() * 2 + col()` will evaluate to 6.

## Function Calls

A function expression consists of an identifier and an optional list of expressions enclosed in parentheses and separated by commas. The value of the expression is the result of applying the function to the arguments passed in as expressions. The arguments are evaluated from left to right before the function is called. For more detail, see section 3.5.

## Range Expressions

Range expressions are used to select part or all of a range. A range expression consists of a bare identifier, a bare range literal, or an expression and a selector. If a range expression has exactly 1 row and 1 column, the value of the expression is the value of the single cell of the range. If it has more than 1 row or more than 1 column, the value of the expression is the selected range. If the range has zero or fewer rows or zero or fewer columns, the value of the expression is `empty`. If a range expression with a selector would access a row index or column index greater than the number of rows or columns of the range, or a negative row or column index, the value of the expression is `empty`.

## Slices

A slice consists of an optional integer literal or expression `start`, a colon, and an optional integer literal or expression `end`, or a single integer literal or expression `index`. If `start` is omitted, it defaults to 0. If `end` is omitted, it defaults to the length of the dimension. A single `index` with no colon is equivalent to `index:index+1`. Enclosing `start` or `end` in square brackets is equivalent to the expression `row() + start` or `row() + end`, for a row slice, or `column() + start` or `column() + end` for a column slice. The slice includes `start` and excludes `end`, so the length of a slice is `end - start`. A negative value is interpreted as the length of the dimension minus the value. As mentioned above, the value of a range that is not 1 by 1 is a range, but the value of a 1 by 1 range is essentially dereferenced to the result of the cell formula.

## Selections

A selection expression consists of an expression and a pair of slices separated by a comma and enclosed in square brackets, i.e. `[row_slice, column_slice]`. If one of the dimensions of the range has length 1, the comma and the slice for that dimension can be omitted. If the comma is present but a slice is omitted, that slice defaults to `[0]` for a slice corresponding to a dimension of length greater than one, or `0` for a slice corresponding to a dimension of length one.

## Corresponding Cell

A very common selection to make is the cell in the "corresponding location" of a different variable. Since this case is so common, `#var` is syntactic sugar for `var[,]`. As a result, if `var` has more than column and more than one row, `#var` is equivalent to `var[row()],column()]`. If `var` has multiple rows and one column, it is equivalent to `var[row(),0]`. If `var` has one row and multiple columns, it is equivalent to `var[0,column()]`; and if `var` has one row and one column, it is equal to `var[0,0]`.

## Selection Examples

```
1 selection_examples() {
2   foo[0,2] /* This evaluates to the cell value in the first row and third column. */
3   foo[0,:] /* Evaluates to the range of cells in the first row of foo. */
4   foo[:,2] /* Evaluates to the range of cells in the third column of foo. */
5   foo[:,[1]] /* The internal brackets denote RELATIVE notation.
6   In this case, 1 column right of the column of the left-hand-side cell. */
7
8   foo[3,] /* Equivalent to foo[3,[0]] if foo has more than one column
9   or foo[3,0] if foo has one column */
10
11  foo[5:, 7:] /* All cells starting from the 6th row and 8th column to the bottom
12             right */
13  foo[[1]:[2], 0:[7]]
14  /* Selects the rows between the 1st and 2nd row after LHS row, and
15     all the columns up to the 7th column to the right of the LHS column */
16
17  /* In this example, each cell of bar would be equal to the cell
18     * in foo in the equivalent location plus 1. */
19  [5,5] foo;
20  [5,5] bar := #foo + 1; // #foo = foo[[0],[0]]
21
22  /* In this example, bar would be a 3x5 range where in each row,
23     * the value in bar is equal to the value in foo in the same column.
24     * In other words, each row of bar would be a copy of foo. */
25  [1,5] foo; // foo has 1 row, 5 columns
26  [3,5] bar := #foo; // #foo = foo[0,[0]]
27
28  /* In this example, the values of baz would be
29     * 11, 12, 13 in the first row;
30     * 21, 22, 23 in the second row;
31     * 31, 32, 33 in the third row. */
32  foo := {1,2,3}; // 1 row, 3 columns
33  bar := {10;20;30}; // 3 rows, 1 column
34  [3,3] baz := #foo + #bar; // Equivalent to foo[0,[0]] + bar[[0],0]
35 }
```

## Precedence Expressions

A precedence expression is used to force the evaluation of one expression before another, when that order of operation is required for functions with side-effects. It consists of an expression `prec-expr`, the precedence operator `->`, and an expression `succ-expr`. The value of the expression is `succ-expr`, but the value of `prec-expr` will be calculated first and the result ignored. All functions written purely in Extend are free of side effects. However, some of the external functions provided by the standard library, such as for file I/O and plotting, do have side effects. The precedence operator has the second-lowest grammatical precedence of all operators, higher only than the ternary operator.

## 3.5 Functions

The bulk of an Extend program consists of functions. Although Extend has some features, such as immutability and lazy evaluation, that are inspired by functional languages, its functions are not *first class objects*. By default, the standard library is automatically compiled and linked with a program, but there are no functions built into the language itself.

### Format

As in most programming languages, the header of the function declares the parameters it accepts. The body of the function consists of an optional set of variable declarations and formula assignments, which can occur in any order, and a return statement, which must be the last statement in the function body. All variable declarations and formula assignments, in addition to the return statement, must be terminated by a semicolon. This very simple function returns whatever value is passed into it:

```
1  foo(arg) {  
2      return arg;  
3  }
```

### Variable Declarations

A variable declaration associates an identifier with a range of cells of the specified dimensions, which are listed in square brackets before the identifier. For convenience, if the square brackets and dimensions are omitted, the identifier will be associated with a single cell. In addition, multiple identifiers, separated by commas, can be listed after the dimensions; all of these identifiers will be separate ranges, but with equal dimension sizes. The dimensions can be specified as any valid expression that evaluates to a Number, which will be rounded to the nearest signed 32-bit integer. If either dimension is zero or negative, or if the expression does not evaluate to a Number, a runtime error causing the program to halt will occur.

```
1  [2, 5] foo; // Declares foo as a range with 2 rows and 5 columns  
2  [m, n] bar; // Declares bar as a range with m rows and n columns  
3  [3, 3] ham, eggs, spam; // Declares ham, eggs and spam as distinct 3x3 ranges  
4  baz; // Declares baz as a single cell
```

### Formula Assignment

A formula assignment assigns an expression to a subset of the cells of a variable. Unlike most imperative languages, this expression is not immediately evaluated, but is instead only evaluated if and when it is needed to calculate the return value of the function. A formula assignment consists of an identifier, an optional pair of slices enclosed in square brackets specifying the subset of the cells that the assignment applies to, an `=`, and an expression, followed by a semicolon. As with the expressions specifying the dimensions of a range, these slices specifying the cell subset can contain arbitrary expressions, as long as the expression taken as a

whole evaluates to a Number, which will be rounded to the nearest signed 32-bit integer. Negative numbers are legal in these slices, and correspond to (dimension length + value).

```
1 [5, 2] foo, bar, baz; // Declares foo, bar, and baz as distinct 5x2 ranges
2 foo[0,0] = 42; // Assigns the expression 42 to the first cell of the first row of foo
3 foo[0,1] = foo[0,0] * 2; // Assigns (foo[0,0] * 2) to the 2nd cell of the 1st row of
   foo
4 bar = 3.14159; // Assigns pi to every cell of every row of bar
5 baz[1:-1,0:1] = 2.71828; // Assigns e to cells (1,0) through (3,1), inclusive, of baz
6
7 /* The next line assigns foo[[-1],0] + 2 to every cell in
8    both columns of foo, besides the first row */
9 foo[1:,: ] = foo[[-1],0] + 2;
```

The last line of the source snippet above demonstrates the idiomatic Extend way of simulating an imperative language's loop; `foo[4,0]` would evaluate to  $42+2+2+2+2 = 50$  and `foo[4,1]` would evaluate to  $(42*2)+2+2+2+2 = 92$ .

### Combined Variable Declaration and Formula Assignment

For convenience, a variable declaration and a formula assignment to all cells of that variable can be combined on a single line by inserting a `:=` and an expression after the identifier. Multiple variables and assignments, separated by commas, can be declared on a single line as well. All global variables must be defined using the combined declaration and formula assignment syntax.

```
1 /* Creates two 2x2 ranges; every cell of foo evaluates to 1 and every cell of
2    bar evaluates to 2. */
3 [2,2] foo := 1, bar := 2;
```

### Formula Assignment Errors

If the developer writes code in such a way that more than one formula applies to a cell, a runtime error will occur if the cell's value is required to compute the return expression. If there is no formula assigned to a cell, the cell will evaluate to `empty`.

### Parameter Declarations

Parameters can be declared with or without dimensions. If dimensions are declared, they can either be specified as integer literals or as identifiers. If a dimension is specified as an integer literal, the program will verify the dimension of the argument before beginning to evaluate the return expression; if it does not match, a runtime error will occur causing the program to halt. If it is specified as an identifier, that variable will contain the dimension size and will be available inside the function body. If the same identifier is repeated in the function declaration, the program will verify that every parameter dimension with that identifier has equal dimension size; if they differ, a runtime error will occur causing the program to halt. A few examples follow:

```
1 number_of_cells([m,n] arg) {
2   return m*n; // m and n are initialized with the dimensions of arg
3 }
4
5 die_unless_primitive([1,1] arg) {
6   return 0; // If arg is not a primitive value, a runtime error will occur
7 }
8
9 num_cells_if_column_vector([m,1] arg) {
10  // If arg has one column, return number of cells; otherwise runtime error
```

```

11     return m;
12 }
13
14 die_unless_square([m,m] arg) {
15     return 0; // Runtime error if number of rows != number of columns
16 }
17
18 num_cells_if_same_size([m,n] arg1, [m,n] arg2) {
19     // If arguments are the same size, return # of cells, otherwise runtime error
20     return m*n;
21 }
22
23 main(args) {
24     [3,4] foo;
25     [3,5] bar;
26     return print_endline(num_cells_if_same_size(foo,bar));
27 }

```

## Application on Ranges

Extend gives the developer the power to easily apply operations in a functional style on ranges. For example, the following function performs cell wise addition:

```

1 foo([m,n] arg1, [m,n] arg2) {
2     [m,n] bar := #arg1 + #arg2;
3     return bar;
4 }

```

This function normalizes a column vector to have unit norm:

```

1 normalize_column_vector([m,1] arg) {
2     [m,1] squared_lengths := #arg * #arg, normalized := #arg / vector_norm;
3     vector_norm := sqrt(sum(squared_lengths));
4     return normalized;
5 }

```

## Lazy Evaluation and Circular References

All cell values and variable dimensions are evaluated lazily if and when they are needed to calculate the return expression. Using lazy evaluation ensures that the cell values are calculated in a valid topological sort order and allows for detection of circular references; internally this is accomplished by constructing a function for each formula which is called the first time the cell's value is needed, and marking the cell as "in-progress" once it starts being evaluated and as "complete" once the value has been calculated. The only guarantees the language places on the order of cell evaluation are: (1) It will be a valid topological ordering; (2) In conditional expressions and in short-circuiting operator expressions, only the relevant conditional branches will be evaluated; and (3) In an expression using the precedence operator, the preceding expression will be evaluated before the succeeding expression. A range selection consisting of multiple cells will not cause the constituent cells to be evaluated; however, selection of a single cell will cause that cell's value to be evaluated. If a program is written in such a way as to cause a circular dependency of one cell on another, and the return expression is dependent on that cell's value, a runtime error will occur. For example, in the following function:

```

1 maybeCircular(truth_value) {
2     x := x;
3     return truth_value ? x : 0;

```

```

4  }
5
6  main(args) {
7      foo :=
8          print_endline("To be or not to be?") ->
9          print_endline("Enter \"Not to be\" to attempt to evaluate a circular reference.")
            ->
10         readline(STDIN);
11
12     return
13         maybeCircular(foo == "Not to be" || foo == "\"Not to be\"") ->
14         print_endline("Good thing I didn't look at the value of x.");
15 }

```

A runtime error will occur if `maybeCircular(1)` is called; but if `maybeCircular(0)` is called, the function will simply return 0.

## External Libraries

Using the following library declaration:

```

1 extern "mylib.o" {
2     foo(arg1, arg2);
3     bar();
4 }

```

will make the functions `foo` (taking two arguments) and `bar` (taking zero arguments) available within `Extend`. In LLVM, the compiler will declare external functions `extend_foo` and `extend_bar` as functions of two and zero arguments respectively. All arguments must have the type `value_p`, and the function must have return type `value_p`, declared in the `Extend` standard library header file. In other words, the C file compiled to generate the library must have defined:

```

1 value_p extend_foo(value_p arg1, value_p arg2) {
2     /* function body here; */
3 }
4
5 value_p extend_bar() {
6     /* function body here; */
7 }

```

## 3.6 Standard Library Reference

### 3.7 Example Program

```

1 import "./samples/stdlib.xtnd";
2
3 main([1,n] args) {
4     /* Get a working copy */
5     return 0;
6 }

```

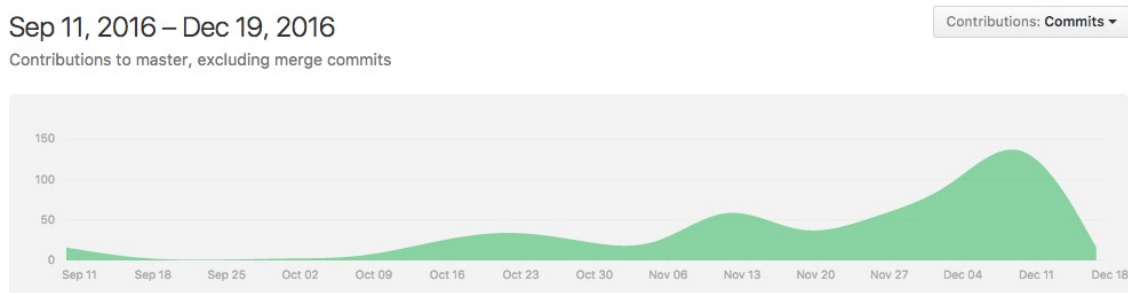
## 4. Project Plan

### 4.1 Meetings

Our goals were outlined by weekly meetings. We regularly met with Jacob Graff, our advisor throughout the development of Extend. Jacob served as a sounding board whenever Extend’s fundamental design philosophy was debated, and as a guide as we determined whether we were on track. We used any leftover time on those days to set goals for the upcoming week and pair program if time permitted.

Our team also met weekly on Fridays to further discuss the progression of Extend. In the first half of the semester, the discussions were primarily philosophical, as decisions had to be made about the language grammar and behavior of certain Extend artifacts prior to development. In the second half, time was devoted to ironing out the development timeline, discussing bugs, and making compiler implementation decisions.

### 4.2 Development Workflow



### Github & Travis CI

Our development and documentation were all done entirely through version control to maximize independent productivity. New features were introduced to the master branch through pull requests, and the team used this as a platform to peer review code to maximize code quality before such features entered production.

An important aspect of development for us was continuous integration. Each pull request we made triggered a Travis build, which kept us informed regarding unexpected hiccups that sometimes arose during development. Travis CI ensured that new features were implemented with protecting the code base in mind, and provided quick visibility as to whether a new feature would break the existing build. Any changeset to the master branch must:

1. Pass Travis CI.
2. Be approved by another member of the team.



3. Be up to date with the master branch.

## 4.3 Project Feature Completion Timeline

Over the semester, we implemented our compiler from front end to back end, incorporating test cases throughout the way. Below are timestamps of our project progress throughout the semester.

1. Scanner (Early October)
2. Parser and JSON output (Mid October)
3. Finalize Language Semantics (Late October)
4. Implement interpreter (Mostly feature complete by early November)
5. Most transformations done, compile Hello World (Mid November)
6. Finalize test suite (End November)
7. Compile function calls, finish transformations (Early December)
8. Compile references to variables (Early December)
9. Feature complete compiler (December 15)
10. Presentation to Professor Edwards (December 19)

## Style Guide

None of the team members had any prior experience with Ocaml. Fundamentally we were developing a certain style in the process of creating the project. A few style choices were clear soon after starting to develop in Ocaml:

- Avoid deep nesting of functions
- Instead build better abstraction and reuse functions
- Use **let ... in** instead of **and**. While this creates a lot of closures, it helped us to develop quicker by not needing to restructure code for changes
- Use underscore for values you won't use any further. Llvm code generation inherently creates a lot of values where the return value is of no use. Therefore mark those return values with an underscore, since it hides the warning.
- Indent with spaces, not tabs. Indent by 2 for each level of nesting.
- Make intentions clear by naming return values, not by naming LLVM intermediate values.

These few rules helped us to control our code very well.

Further we were developing our runtime in C. We applied the following style rules:

- Indent with tabs.
- Stick to C99.
- Use **value\_p** for user facing functions.
- Make sure to exit gracefully.

## 4.4 Language Evolution

The language we delivered ended up surprisingly close to our initial proposal. The biggest change was to allow strings and ranges as value types, which made the language immeasurably better. Initially, we wanted to only have Numbers; but allowing cells to contain ranges (composite values) and not just primitive values makes it a much more useful language. Otherwise, the syntax and semantics are very close to what was in our original proposal.

Our initial plan was to precalculate the dependencies among cells as best as possible at compile time and generate code accordingly. However, it quickly became clear that the language was better with runtime-determined cell dependencies and we therefore had to give up on a precomputed graph. We didn't have time in this class to implement an explicit stack as opposed to using recursion, but this could be overcome if it had to be.

One minor change was to eliminate the dimension signature for functions. As we played around with the language in the interpreter, it became clear that we weren't using them and it wasn't obvious why we would; they were dropped as a result.

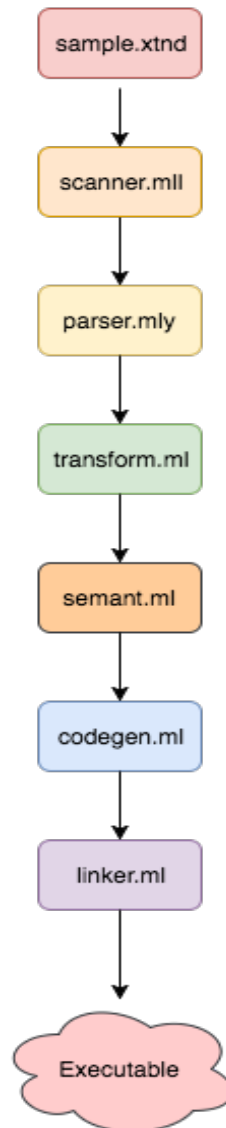
### The Interpreter

Mainly because we implemented a declarative language, we built a working interpreter fairly early in the process to make sure we understood how to actually compile our language. Having it allowed us to test the language semantics, run example Extend programs, and make language decisions at an earlier stage. It also helped us benchmark the success of our compiler by comparing the number of testcases passed by both. A lot of the expressive power of Extend comes from the selection / slicing operator and a surprisingly high percentage of code in the compiler (20% of the C runtime!) is devoted to handling selections. Having the details worked out in the interpreter gave us a road map that made the corresponding LLVM code generation much more straightforward. In addition, it was not obvious before having a working interpreter that we would need to have a scope (closure) object in order to allow recursion; if we had missed this important detail, we likely would have needed to make drastic code or language changes late in the process. Finally, it gave us confidence that the various transformations we performed on the source produced correct results.

## 4.5 Team Member Responsibilities

Team Member	Responsibilities	GitHub Profile
Jared Samet	design philosophy, semantic transformations, code generation	<a href="#">oracleofnj</a>
Nigel Schuster	development protocol, code generation, scripting	<a href="#">Neitsch</a>
Ishaan Kolluri	initial LRM, Final Report, regression tests, stdlib functions, scripting	<a href="#">ishaankolluri</a>
Kevin Ye	initial scanner, regression tests, stdlib functions	<a href="#">kevinye1</a>

## 5. Extend's Internal Architecture



## 5.1 The Extend Compiler

The Extend compilation process consists of several source files, each of which performs a different function in the compilation pipeline.

- `scanner.mll`: OCamllex scanner - consumes tokens.
- `parser.mly`: OCamllyacc parser - represents the Extend grammar.
- `ast.ml`: Abstract Syntax Tree, created from the output of the parser and representing the structure of an Extend program.
- `transform.ml`: Performs syntactic desugaring for easier compilation.
- `semant.ml`: Analyzes the semantics of the program to ensure that the program adheres to the rules of the language.
- `codegen.ml`: The LLVM IR code generator.
- `linker.ml`: Calls intermediary compilation steps on the generated `.ll`, including external functions if needed.

### The Scanner

The function of `scanner.mll` is to parse a text stream into various tokens to be used in an Extend program. Only the tokens that are valid in Extend are to be given to the parser; all others will return a syntax error marked by the line and character number.

### The Parser and Abstract Syntax Tree

The parser converts the tokens read by the scanner into a syntax tree deemed acceptable grammar within the Extend Language. This is converted into an Abstract Syntax Tree, which has nodes that can be consumed by the back end of the Extend compiler.

### The Transformer

The transformer is the first step in converting the AST into LLVM code. It takes the AST and reduces its breadth. This step is done to preserve the convenience for the user, but reduces the complexity for the actual compile step. A large number of internal variables are created in the process.

This is how the user declares a variable.

```
1 [2,2] foo;
```

This is how the transformer desugars the same code.

```
1 rows_of_foo := 2;  
2 cols_of_foo := 2;  
3 [rows_of_foo, cols_of_foo] foo;
```

A similar transformation is performed on formula assignments:

```
1 // Before Transformation:  
2 foo[g(x):4,3+3] = "Couldn't you have stuck to integers?";  
3  
4 // After Transformation:  
5 start_row := g(x);
```

```

6   end_row    := 4;
7   start_col  := 3+3;
8   foo[start_row:end_row,start_col] = "Couldn't you have stuck to integers?";

```

Every expression on the left hand side before or after a comma or colon will become an internal temporary variable in the desugaring process. Internal variables are also created for the return expression and for any size assertions induced by the function signature:

```

1   // Before Transformation:
2   foo([m,n] arg1, [m, 1] arg2) {
3       return m*n;
4   }
5
6   // After Transformation:
7   foo(arg1, arg2) {
8       m := numRows(arg1);
9       n := numCols(arg1);
10      asserts := (m == numRows(arg2)) && (1 == numCols(arg2));
11      return_value := m*n;
12      return return_value;
13  }

```

In addition to generating temporary variables, Extend also transforms `&&`, `||`, and `switch` into ternary conditionals to enable short-circuiting. Finally, the transformer performs some semantic analysis to ensure that there are no duplicate variables within a function, and no duplicate functions within a program.

## The Semantic Analyzer

The semantic analyzer consumes the reduced AST. It ensures that Extend functions, variables, expressions, and more are being used properly at compile time, and throws flavorful exceptions to the user so that they may better understand why their program was illegal. In Extend, there are no real type errors involving expressions on the right-hand-side of a formula, as we attempt to degrade many gracefully. There are type errors possible on the left-hand side, but since they are assigned dynamically, very few can be determined at compile time. For function calls, the semantic analyzer ensures that the function exists and is called with the right number of arguments; and for variables, the analysis checks that the identifier refers to a real variables within the appropriate scope.

## The Code Generator

Once the Extend AST passes semantic analysis, the code generator turns the reduced AST into LLVM code. Since the variable evaluation approach of Extend is not imperative, this process is fairly elaborate. There is one function created per formula, which is available to be called if the value of a cell with that formula is needed; and there is one function created per Extend function, which initializes a scope object with a collection of blueprints for all the local variables of that function. In its most basic form, each blueprint has a reference to one or more formulas that calculate the value of the variable. The section on the runtime goes into more detail on how this architecture is used.

## The Linker

If successful LLVM IR is generated, the linker will adopt the role of building an executable object from the `.ll` file. This includes compiling it to an object file and linking the runtime environment along with other imported libraries.

## 5.2 Extend Runtime

Extend's cell values are lazily evaluated, which means they need to be implemented using function pointers. For each function that the Extend developer writes, the corresponding LLVM function that is generated is essentially identical: allocate a scope object for that function call, initialize that object with the appropriate set of variable definitions and the function arguments to that scope object, and then evaluate the variables corresponding to the size assertion and the return expression for that function. All of the "individualized" code lives in what we refer to as the formula-functions; for each distinct formula, the compiler generates a corresponding function that can be called when the corresponding cell's value is needed. Each formula-function shares the same signature: the arguments are a pointer to a scope and the row and column number of the cell being evaluated, and the return value is a pointer to a value struct (which holds the type and contents of the value.)

The two main functions of our C runtime, therefore, are `instantiate_variable()`, which looks at the variable definition "blueprint" and calculates the actual dimensions of the variable for that particular function call, and calculates the actual range of cells to which each formula applies; and `getVal()`, which determines if a particular cell value has already been calculated or not, and calls the appropriate formula-function if not.

Before actually calling the main Extend entry point, our executable initializes a global array with the appropriate variable definitions for each function. When an Extend function is called, it simply copies the appropriate pointer into that array into its scope object.

Leaving aside the variables introduced by the transformation step, this Extend function:

```
1  foo() {
2      x := 1;
3      return x;
4  }
```

would result in the LLVM equivalent of the following pseudocode (not written in any actual language) being generated:

```
1  value_p foo() {
2      scope = new ExtendScope;
3
4      // Load the appropriate set of definitions for foo
5      scope->defns = global_definitions[42];
6
7      // Create an array of pointers to variable instances; one
8      // pointer per variable. Only one variable in this function
9      scope->insts = new var_instance* [1];
10
11     // getVar calls instantiate_var if that instance pointer is still NULL,
12     // or just returns the pointer if it's already been instantiated.
13     // The instantiated variable keeps a copy of the pointer to its scope
14     var_instance *return_variable = getVar(scope, 0);
15
16     // Get the value of cell [0,0] of return_variable
17     return getVal(return_variable, 0, 0);
18 }
```

Since the newly initialized scope object will hold all NULL pointers for the instances, `getVar()` will end up calling `instantiate_variable`, which will determine that `x` has 1 row and 1 column; there is only a single formula for `x`, applying to all cells of `x`; and that that formula corresponds to the function pointer indicated in the variable definition. When `getVal` is called, the value pointer for the `[0,0]`th cell will similarly be NULL. As a result, `getVal()` will determine the function pointer for the appropriate formula and then call it, supplying as arguments a pointer to the scope and `(0,0)` for the row and column.

The actual C structures used are listed below:

```

1
2 // Each formula-function has the following signature:
3 typedef value_p (*FormulaFP) (struct ExtendScope *scope, int row, int col);
4
5 // This structure tells the runtime how to actually calculate the range of
6 // cells to which each formula applies.
7 struct ExtendFormula {
8     /* These 10 variables correspond to formula_row_start through formula_col_end,
9      * where char singleRow/Col are true if formula_row_end is None */
10    char fromFirstRow;
11    int rowStart_varnum;
12    char toLastRow;
13    int rowEnd_varnum;
14    char fromFirstCol;
15    int colStart_varnum;
16    char toLastCol;
17    int colEnd_varnum;
18
19    char isSingleRow;
20    char isSingleCol;
21
22    FormulaFP formula;
23 };
24
25 // For a particular variable instance, this structure holds the results
26 // of the calculations for each formula.
27 struct ResolvedFormula {
28     int rowStart, rowEnd, colStart, colEnd;
29     FormulaFP formula;
30 };
31
32 struct var_defn {
33     /* This is like a class definition – for every declared variable in the
34      * Extend source, there should be one instance of these per compiled program.
35      * They should just live in the global program storage.
36      * It corresponds to Ast.variable */
37     int rows_varnum;
38     int cols_varnum;
39     int numFormulas;
40     struct ExtendFormula *formulas;
41     char isOneByOne;
42     char *name;
43 };
44
45 struct var_instance {
46     /* This is an actual instance of a variable – we get one of these
47      * per variable per time a function is called (assuming the contents
48      * of the variable get examined. */
49     int rows, cols;
50     int numFormulas;
51     struct ResolvedFormula *formulas;
52     struct ExtendScope *closure;
53     value_p *values;
54     char *status;
55     char *name;
56 };

```

```
57
58 // One scope object gets created per Extend function call
59 struct ExtendScope {
60     struct var_defn *defns;
61     struct var_instance **vars;
62     int numVars;
63     int refcount;
64     value_p *functionParams;
65 };
```



## 6. Testing

Due to Extend being a large undertaking, we took steps to ensure that all features were working as the design of the language intended.

This was done through implementing test cases that isolated specific aspects of the Extend language to ensure that each feature worked correctly. For basic components, we wrote a plethora of tests to illustrate functionality. For undertakings that required more debate on the design of the language, other tests were created and modified throughout development.

### 6.1 Feature Integration & Testing

Development of new features naturally means that they must be deemed legal by the scanner, parser, semantic analyzer, and code generator. As we developed new features, the process was roughly as follows:

1. Write a simple test that illustrated the feature to test.
2. Write the expected output of the aforementioned test to a text file.
3. Confirm that the scanner consumes the tokens related to the feature.
4. Confirm that the parser grammar has been adjusted to accomodate the new feature.
5. Confirm that the semantic analyzer and transformer can properly identify and check the new feature code.
6. Confirm that code generation generates the appropriate LLVM IR for the new features - such as allocating memory, building calls, and more.
7. Ensure that the test written can write its output to `stdout`, to be compared with expected output.
8. Compile and test the code to ensure that the code has worked to the team's expectations.

Earlier in the development process, we tested the front end of our compiler by JSON-ifying the abstract syntax tree, printing it, and examinining it. As we settled into full-fledged development, we would test with a full-feature regression test suite. Later in the semester, JSON-ifying still proved to be useful, as it gave us the option to print debug statements if needed.

### 6.2 Regression Test Suite

Extend's test suite is executable through the `testscript.sh` script at the top level of the project. There are over 100 integration test files for various features of the Extend language, and a corresponding file with their expected output to `stdout`. This is to ensure that the successful implementation of one feature does not impact that of others.

Regression tests were placed in the `testcases/inputs_regression` directory. Tests that did not pass at the time were placed in the `testcases/inputs` directory. The test script compiles and executes each test, and compares it with the corresponding expected output file, living in the `testcases/expected` directory. Whenever a test passed in `inputs`, it was automatically moved over to `inputs_regression`.

**Note:** We have added a full test listing at the end of this document. Please refer to the chapter titled "Test Listing" for more detail.

## Integration with Travis CI

The aforementioned test suite is run by Travis CI in the event that the Extend compiler is successfully built; otherwise, the build will fail and exit. In our development workflow, checking the logs during build failures sometimes revealed that tests in the regression test suite did not succeed as expected. This integration kept the far-reaching effects of newly introduced features entirely transparent throughout the process.

Using Travis CI allowed us to maintain the working ability of our compiler, as it ensured that every new feature pushed to the master branch would still result in a successful build. This proved to be invaluable when testing the compiler at a macro-level, or providing Jacob, our TA, with up-to-date demonstrations.

## 7. Example Source Programs

Below are two example programs we've implemented in Extend to illustrate some of our language's features and use cases.

### 7.1 maybeCircular

This program illustrates how Extend lazily evaluates. Since we shortcircuit the ternary conditional below, based on what the user inputs, this program will either complete or throw a runtime error.

```
1 maybeCircular(truth_value) {
2   x := x;
3   return truth_value ? x : 0;
4 }
5
6 main(args) {
7   foo :=
8     print_endline("To be or not to be?") ->
9     print_endline("Enter \"Not to be\" to attempt to evaluate a circular reference.")
10    ->
11    readline(STDIN);
12
13   return
14     maybeCircular(foo == "Not to be" || foo == "\"Not to be\"") ->
15     print_endline("Good thing I didn't look at the value of x.");
16 }
```

### 7.2 True Shooting Percentage

This program parses calculates the true shooting efficiency NBA players. It reads in a string from a file, parses it into a variable, and prints and calculates the true shooting percentage for each player based on values in the vector. It additionally prints the player with the highest percentage, and writes the results to a GIF bar chart.

```
1 main(args) {
2   welcome := "NBA True Shooting Percentage\n-----";
3   data := parseString(read(open("tsp_data", "r"), 0));
4
5   // Calculates TSP for each player
6   [10,2] players;
7   players[:,0:1] = data[[0],[0]];
8   players[:,1:] = calculate_tsp(data[[0],1],data[[0],2],data[[0],3]);
9
10  // Calculates which player has the highest TSP
```

```

11 player := highest_tsp(players);
12 [10,1] playerSummary := players[[0],0] + ": " + toString(players[[0],1]);
13
14 return
15   print_endline(welcome) ->
16   print_endline(concatRow(transpose(playerSummary), "\n")) ->
17   print_endline("_____") ->
18   print_endline("The player with the highest True Shooting Percentage is " + player
19     [0,0] + " with a TSP of " + toString(player[0,1]) + "!") ->
19   bar_chart(open("barchart.png", "wb"), transpose(players[:,0]), transpose(players
20    [:,1]));
21
22 calculate_tsp(pts, fga, fta) {
23   tsp := pts / (2.0 * (fga + (0.44 * fta)));
24   return tsp;
25 }
26
27 highest_tsp([m,n] players) {
28   [m,1] tsp_ranking;
29   tsp_ranking[0,0] = players[0,:];
30   tsp_ranking[1:,:] = (players[[0],1] > tsp_ranking[[-1],0][1]) ? players[[0],:] :
31     tsp_ranking[[-1],0];
32   return tsp_ranking[m-1,0];
33 }

```

## 8. Reflection

### 8.1 Ishaan

When working on a long-term project, communication is paramount. Throughout this project, I realized that maximum productivity occurred when the team kept a constant line of communication open regarding language and code design. Additionally, it's important to identify where people can be most productive. If one person is more efficient at a certain task, more progress will be made if they work on similar material. Lastly, as the design of the language evolves over time, it's important to build a system that allows for flexibility, as you never know what may change later in the development process.

### 8.2 Jared

I really enjoyed this project from start to finish. In my former life, I worked in finance and was intimately familiar with Excel's strengths and weaknesses as a result, and as the language guru I tried to incorporate what I thought were the best points of spreadsheets into our language. It was a lot of work, but of the good kind - the appeal of being able to build something and see it in action is what brought me back from finance in the first place. Over a two-day span, our compiler went from not being able to handle "==" to being nearly feature complete; it was an incredible feeling to see its expressive power explode as we successively implemented each additional basic building block of the language.

Things I think we did well: To my eyes, the syntax is concise without being incomprehensibly terse and it is easy to write programs in the language; I'm still impressed by how few lines of code it took to implement the `splitToRange()` function; and I think we essentially delivered what we had in mind when the project began. After being a thorn in my side for weeks, I think we finally implemented literals correctly (initialize once and then do shallow copies when they're actually referenced.) Having a working interpreter very early in the process made it easy to test out the syntax of the language, come up with some test cases, and have a concrete game plan for the actual implementation of the compiler.

Things where we could have done better: We followed the MicroC template a little too closely and it would have been better to implement a separate SAST as opposed to just an AST. Although I am fairly sure we don't allow any semantic errors past, it would have been nice to have the "extra confidence" that a SAST would have given us that all of the symbols would indeed be where they were supposed to be, enforced by the typing. We whiffed on memory management. I was disappointed that we didn't have time to implement an explicit stack instead of using recursion but came to terms with that.

All in all, this was a fantastic experience and I had a great time working with the team!

### 8.3 Nigel

Team projects by its nature are a very unique challenge for a student. Nevertheless these projects are incredibly valuable by providing a more applied experience. Thus, I am glad that I was able to put a lot of effort into this project. Communication proved to be a key element in the project: We had weekly team meetings, meetings with the TA, a chatroom and ad hoc in person discussions. All this helped to bounce

ideas off one another, prioritize well and avoid a mismatch in expectations. Of course some problems are inevitable. Therefore I think one of our key assets was our test suite. At any point in time it allowed us to see the next step ahead - the next thing we want to make work. In the same vein our code review process proved very effective (PR required approval plus passing CI). I admit that at some points in the development process I was slacking off, especially when facing LLVM codegen for the first time. However I am glad, that my team mates motivated me and helped me to get back on track. Summarizing, every project has its issues, but by planning ahead and hard work, we built a surprisingly good and feature complete language that is close to our initial goal.

## 8.4 Kevin

Working on this group project this semester has been a rewarding experience and posed quite the challenge. It was something very new to me and I had trouble at first balancing all my work. But I slowly adapted and got used to it. My takeaway from this experience would have to be learning the importance of communication and having a set structure. One of my biggest problems in life is that I have a hard time asking for help. Mainly because I'm afraid of getting judged for asking a dumb question. But communication is key in any team project I've learned. I could've easily asked my teammates, who were always willing to help, for help on a problem I'm having than spent hours trying to figure it out on my own. And oftentimes, in doing so, I would learn something new, which is great. I had several other classes this semester that also had me doing group projects and I felt like the overall workflow for those group projects weren't as organized as our PLT project. This was simply due to the fact that we set a structure right from the start. We had weekly meetings with our advisor along with weekly meetings with each other and a group chat, which when all combined together kept us on track on everything that needed to be done.

## 9. Extend Code Listing

### 9.1 scanner.mll

```
1  (* jss2272 isk2108 ky2294 *)
2
3  {
4    open Lexing
5    open Parser
6    open String
7
8    exception SyntaxError of string
9    let syntax_error lexbuf = raise (SyntaxError("Invalid character: " ^ Lexing.lexeme
        lexbuf))
10 }
11
12 let digit = ['0'-'9']
13 let exp = 'e' ('+'|'-'|'|'/'|'^')? ['0'-'9']+
14 let flt = (digit)+ ('.' (digit)* exp)?|exp)
15 let id = ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_' ]*
16
17
18 rule token = parse
19   ['\n']          { new_line lexbuf; token lexbuf }
20 | [' ' '\t' '\r'] { token lexbuf } (* Whitespace *)
21 | "/*"            { multiline_comment lexbuf }
22 | "//"           { oneline_comment lexbuf }
23 | '"'            { read_string (Buffer.create 17) lexbuf }
24 | '['            { LSQBRACK }
25 | ']'            { RSQBRACK }
26 | '('            { LPAREN }
27 | ')'            { RPAREN }
28 | '{'            { LBRACE }
29 | '}'            { RBRACE }
30 | ":@"           { GETS }
31 | '='            { ASN }
32 | ':'            { COLON }
33 | ','            { COMMA }
34 | "->"           { PRECEDES }
35 | '?'            { QUESTION }
36 | "=="           { EQ }
37 | "!="           { NOTEQ }
38 | '<'            { LT }
39 | '>'            { GT }
40 | "<="           { LTEQ }
```

```

41 | ">="          { GTEQ }
42 | ';'          { SEMI }
43 | '!'          { LOGNOT }
44 | "&&"          { LOGAND }
45 | "||"         { LOGOR }
46 | '~'          { BITNOT }
47 | '&'          { BITAND }
48 | '|'          { BITOR }
49 | '^'          { BITXOR }
50 | '+'          { PLUS }
51 | '-'          { MINUS }
52 | '*'          { TIMES }
53 | '/'          { DIVIDE }
54 | '%'          { MOD }
55 | "**"          { POWER }
56 | "<<"          { LSHIFT }
57 | ">>"          { RSHIFT }
58 | '#'          { HASH }
59 | "if"         { IF }
60 | "empty"      { EMPTY }
61 | "size"       { SIZE }
62 | "typeof"     { TYPEOF }
63 | "row"        { ROW }
64 | "column"     { COLUMN }
65 | "switch"     { SWITCH }
66 | "case"       { CASE }
67 | "default"    { DEFAULT }
68 | "return"     { RETURN }
69 | "import"     { IMPORT }
70 | "global"     { GLOBAL }
71 | "extern"     { EXTERN }
72 | digit+ as lit { LIT_INT(int_of_string lit) }
73 | flt as lit   { LIT_FLOAT(float_of_string lit) }
74 | id as lit    { ID(lit) }
75 | eof          { EOF }
76 | _           { syntax_error lexbuf }
77
78 and multiline_comment = parse
79   "*/" { token lexbuf }
80 | '\n' { new_line lexbuf; multiline_comment lexbuf }
81 | _    { multiline_comment lexbuf }
82
83 and oneline_comment = parse
84   '\n' { new_line lexbuf; token lexbuf }
85 | _    { oneline_comment lexbuf }
86
87 (* read_string mostly taken from:
88 https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html *)
89 and read_string buf =
90   parse
91   | '"'         { LIT_STRING (Buffer.contents buf) }
92   | '\n'        { new_line lexbuf; Buffer.add_char buf '\n'; read_string buf lexbuf }
93   | '\\\' 'n'    { Buffer.add_char buf '\n'; read_string buf lexbuf }
94   | '\\\' 'r'    { Buffer.add_char buf '\r'; read_string buf lexbuf }
95   | '\\\' 't'    { Buffer.add_char buf '\t'; read_string buf lexbuf }
96   | '\\\' ([^\\\' 'n' 'r' 't'] as lxm)

```



```

97   { Buffer.add_char buf lxm; read_string buf lexbuf }
98   | [^ '\"' '\\']+
99   { Buffer.add_string buf (Lexing.lexeme lexbuf);
100     read_string buf lexbuf
101   }
102   | _ { syntax_error lexbuf }
103   | eof { raise (Failure("unterminated string")) }

```

## 9.2 parser.mly

```

1  /* Ocaml yacc parser for Extend */
2  /* jss2272 ns3158 */
3
4  %{
5  open Ast
6  %}
7
8  %token LSQBRACK RSQBRACK LPAREN RPAREN LBRACE RBRACE HASH
9  %token COLON COMMA QUESTION IF GETS ASN SEMI PRECEDES
10 %token SWITCH CASE DEFAULT SIZE TYPEOF ROW COLUMN
11 %token PLUS MINUS TIMES DIVIDE MOD POWER LSHIFT RSHIFT
12 %token EQ NOTEQ GT LT GTEQ LTEQ
13 %token LOGNOT LOGAND LOGOR
14 %token BITNOT BITXOR BITAND BITOR
15 %token EMPTY RETURN IMPORT GLOBAL EXTERN
16 %token <int> LIT_INT
17 %token <float> LIT_FLOAT
18 %token <string> LIT_STRING
19 %token <string> ID
20 %token EOF
21
22 %right QUESTION
23 %left PRECEDES
24 %left LOGOR
25 %left LOGAND
26 %left EQ NOTEQ LT GT LTEQ GTEQ
27 %left PLUS MINUS BITOR BITXOR
28 %left TIMES DIVIDE MOD LSHIFT RSHIFT BITAND
29 %right POWER
30 %right BITNOT LOGNOT NEG
31 %left LSQBRACK
32
33 %start program
34 %type <Ast.raw_program> program
35
36 %%
37
38 program:
39     program_piece EOF { let (imp, glob, fnc, ext) = $1 in (List.rev imp, List.rev
40       glob, List.rev fnc, List.rev ext) }
41
42 program_piece:
43     /* nothing */ { ([], [], [], []) }
44     | program_piece import { let (imp, glob, fnc, ext) = $1 in ($2 :: imp, glob,
45       fnc, ext) }

```

```

44 | program_piece global      { let (imp, glob, fnc, ext) = $1 in (imp, $2 :: glob,
    fnc, ext) }
45 | program_piece func_decl { let (imp, glob, fnc, ext) = $1 in (imp, glob, $2 ::
    fnc, ext) }
46 | program_piece extern    { let (imp, glob, fnc, ext) = $1 in (imp, glob, fnc, $2
    :: ext) }
47
48 import:
49     IMPORT LIT_STRING SEMI {$2}
50
51 global:
52     GLOBAL varinit {$2}
53
54 extern:
55     EXTERN LIT_STRING LBRACE opt_extern_list RBRACE {(Library($2, $4))}
56
57 opt_extern_list:
58     /* nothing */ { [] }
59 | extern_list { List.rev $1 }
60
61 extern_list:
62     extern_fn { [$1] }
63 | extern_list extern_fn { $2 :: $1 }
64
65 extern_fn:
66     ID LPAREN func_param_list RPAREN SEMI
67     { {
68         extern_fn_name = $1;
69         extern_fn_params = $3;
70         extern_fn_libname = "";
71         extern_ret_val = (None, None);
72     } }
73
74 func_decl:
75     ID LPAREN func_param_list RPAREN LBRACE opt_stmt_list ret_stmt RBRACE
76     { {
77         name = $1;
78         params = $3;
79         body = $6;
80         raw_asserts = [];
81         ret_val = ((None, None), $7)
82     } }
83
84 opt_stmt_list:
85     /* nothing */ { [] }
86 | stmt_list { List.rev $1 }
87
88 stmt_list:
89     stmt { [$1] }
90 | stmt_list stmt { $2 :: $1 }
91
92 stmt:
93     varinit { $1 } | assign { $1 }
94
95 ret_stmt:
96     RETURN expr SEMI {$2}

```

```

97
98 varinit:
99     var_list SEMI { Varinit((None, None), List.rev $1) }
100 | dim var_list SEMI { Varinit($1, List.rev $2) }
101
102 var_list:
103     ID varassign { [ ($1, $2)] }
104 | var_list COMMA ID varassign { ($3, $4) :: $1}
105
106 varassign:
107     /* nothing */ { None }
108 | GETS expr { Some $2 }
109
110 assign:
111     ID lhs_sel ASN expr SEMI { Assign($1, $2, Some $4) }
112
113 expr:
114     expr rhs_sel      { Selection($1, $2) }
115 | HASH ID             { Selection(Id($2), (None, None)) }
116 | op_expr             { $1 }
117 | ternary_expr        { $1 }
118 | switch_expr         { $1 }
119 | func_expr           { $1 }
120 | range_expr          { $1 }
121 | expr PRECEDES expr  { Precedence($1, $3) }
122 | LPAREN expr RPAREN { $2 }
123 | ID                  { Id($1) }
124 | LIT_INT             { LitInt($1) }
125 | LIT_FLOAT           { LitFlt($1) }
126 | LIT_STRING          { LitString($1) }
127 | EMPTY              { Empty }
128
129 op_expr:
130     expr PLUS expr     { BinOp($1, Plus, $3) }
131 | expr MINUS expr      { BinOp($1, Minus, $3) }
132 | expr TIMES expr      { BinOp($1, Times, $3) }
133 | expr DIVIDE expr     { BinOp($1, Divide, $3) }
134 | expr MOD expr        { BinOp($1, Mod, $3) }
135 | expr POWER expr      { BinOp($1, Pow, $3) }
136 | expr LSHIFT expr     { BinOp($1, LShift, $3) }
137 | expr RSHIFT expr     { BinOp($1, RShift, $3) }
138 | expr LOGAND expr     { BinOp($1, LogAnd, $3) }
139 | expr LOGOR expr      { BinOp($1, LogOr, $3) }
140 | expr BITXOR expr     { BinOp($1, BitXor, $3) }
141 | expr BITAND expr     { BinOp($1, BitAnd, $3) }
142 | expr BITOR expr      { BinOp($1, BitOr, $3) }
143 | expr EQ expr         { BinOp($1, Eq, $3) }
144 | expr NOTEQ expr      { UnOp(LogNot, (BinOp($1, Eq, $3))) }
145 | expr GT expr         { BinOp($1, Gt, $3) }
146 | expr LT expr         { BinOp($1, Lt, $3) }
147 | expr GTEQ expr       { BinOp($1, GtEq, $3) }
148 | expr LTEQ expr       { BinOp($1, LtEq, $3) }
149 | SIZE LPAREN expr RPAREN { UnOp(OfSize, $3) }
150 | TYPEOF LPAREN expr RPAREN { UnOp(.TypeOf, $3) }
151 | ROW LPAREN RPAREN      { UnOp(Row, Empty) }
152 | COLUMN LPAREN RPAREN  { UnOp(Column, Empty) }

```

```

153 | MINUS expr %prec NEG      { UnOp(Neg, $2) }
154 | LOGNOT expr              { UnOp(LogNot, $2) }
155 | BITNOT expr              { UnOp(BitNot, $2) }
156
157 ternary_expr:
158     IF LPAREN expr COMMA expr COMMA expr RPAREN { Ternary($3, $5, $7) }
159 | expr QUESTION expr COLON expr %prec QUESTION { Ternary($1, $3, $5) }
160
161 switch_expr:
162     SWITCH LPAREN switch_cond RPAREN LBRACE default_case_list RBRACE { Switch($3, fst
163     $6, snd $6) }
164 | SWITCH LBRACE default_case_list RBRACE { Switch(None, fst $3, snd $3) }
165
166 switch_cond:
167     /* nothing */ { None }
168 | expr { Some $1 }
169
170 default_case_list:
171     case_list {(List.rev $1, Empty)}
172 | case_list default_expr {(List.rev $1, $2)}
173
174 case_list:
175     case_stmt { [$1] }
176 | case_list case_stmt { $2 :: $1 }
177
178 case_stmt:
179     CASE case_expr_list COLON expr SEMI { (List.rev $2, $4) }
180
181 default_expr:
182     DEFAULT COLON expr SEMI { $3 }
183
184 case_expr_list:
185     expr { [$1] }
186 | case_expr_list COMMA expr { $3 :: $1 }
187
188 func_expr:
189     ID LPAREN opt_arg_list RPAREN { Call($1, $3) }
190
191 range_expr:
192     LBRACE row_list RBRACE { allow_range_literal (LitRange(List.rev $2)) }
193
194 row_list:
195     col_list {[List.rev $1]}
196 | row_list SEMI col_list {List.rev $3 :: $1}
197
198 col_list:
199     expr {[ $1]}
200 | col_list COMMA expr { $3 :: $1}
201
202 opt_arg_list:
203     /* nothing */ { [] }
204 | arg_list { List.rev $1 }
205
206 arg_list:
207     expr {[ $1]}
208 | arg_list COMMA expr { $3 :: $1}

```

```

208
209 lhs_sel:
210     /* nothing */ { (None, None) }
211 /* commented out: LSQBRACK lslice RSQBRACK { (Some $2, None) } */
212 | LSQBRACK lslice COMMA lslice RSQBRACK { (Some $2, Some $4) }
213
214 rhs_sel:
215     LSQBRACK rslice RSQBRACK { (Some $2, None) }
216 | LSQBRACK rslice COMMA rslice RSQBRACK { (Some $2, Some $4) }
217
218 lslice:
219 /* commented out: nothing production { (None, None) } */
220     lslice_val { (Some $1, None) }
221 | lslice_val COLON lslice_val { (Some $1, Some $3) }
222 | lslice_val COLON { (Some $1, Some DimensionEnd) }
223 | COLON lslice_val { (Some DimensionStart, Some $2) }
224 | COLON { (Some DimensionStart, Some DimensionEnd) }
225
226 rslice:
227     /* nothing */ { (None, None) }
228 | rslice_val { (Some $1, None) }
229 | rslice_val COLON rslice_val { (Some $1, Some $3) }
230 | rslice_val COLON { (Some $1, Some DimensionEnd) }
231 | COLON rslice_val { (Some DimensionStart, Some $2) }
232 | COLON { (Some DimensionStart, Some DimensionEnd) }
233
234 lslice_val:
235     expr { Abs($1) }
236
237 rslice_val:
238     expr { Abs($1) }
239 | LSQBRACK expr RSQBRACK { Rel($2) }
240
241 func_param_list:
242     /* nothing */ { [] }
243 | func_param_int_list { List.rev $1 }
244
245 func_param_int_list:
246     func_sin_param { [$1] }
247 | func_param_int_list COMMA func_sin_param { $3 :: $1 }
248
249 func_sin_param:
250     ID { ((None, None), $1) }
251 | dim ID { ($1, $2) }
252
253 dim:
254     LSQBRACK expr RSQBRACK { (Some $2, None) }
255 | LSQBRACK expr COMMA expr RSQBRACK { (Some $2, Some $4) }

```

### 9.3 ast.ml

```

1 (* jss2272 *)
2 type op      = Plus | Minus | Times | Divide | Mod | Pow |
3             LShift | RShift | BitOr | BitAnd | BitXor |
4             Eq | Gt | GtEq | Lt | LtEq | LogAnd | LogOr

```

```

5 type unop      = Neg | LogNot | BitNot | SizeOf | TypeOf | Row | Column | Truthy
6
7 type expr      = LitInt of int |
8                 LitFlt of float |
9                 LitString of string |
10                LitRange of (expr list) list |
11                Id of string |
12                Empty |
13                BinOp of expr * op * expr |
14                UnOp of unop * expr |
15                Ternary of expr * expr * expr |
16                Switch of expr option * case list * expr |
17                Call of string * expr list |
18                Selection of expr * sel |
19                ReducedTernary of string * string * string |
20                Precedence of expr * expr
21 and index      = Abs of expr |
22                Rel of expr |
23                DimensionStart |
24                DimensionEnd
25 and slice      = index option * index option
26 and sel        = slice option * slice option
27 and case       = expr list * expr
28
29 type dim       = expr option * expr option
30 type var       = dim * string
31 type assign    = string * sel * expr option
32 type init      = string * expr option
33 type stmt      = Assign of assign |
34                Varinit of dim * init list
35
36 type raw_func = {
37     name: string;
38     params: var list;
39     body: stmt list;
40     raw_asserts: expr list;
41     ret_val: dim * expr;
42 }
43
44 type extern_func = {
45     extern_fn_name: string;
46     extern_fn_params: var list;
47     extern_fn_libname: string;
48     extern_ret_val: dim;
49 }
50
51 type library    = Library of string * extern_func list
52 type raw_program = string list * stmt list * raw_func list * library list
53
54 (* Desugared types below *)
55 module StringMap = Map.Make(String)
56 type formula    = {
57     formula_row_start: index;
58     formula_row_end: index option;
59     formula_col_start: index;
60     formula_col_end: index option;

```

```

61   formula_expr: expr;
62 }
63
64 type dim_expr = DimOneByOne
65   | DimId of string
66
67 type variable = {
68   var_rows: dim_expr;
69   var_cols: dim_expr;
70   var_formulas: formula list;
71 }
72
73 type func_decl = {
74   func_params: var list;
75   func_body: variable StringMap.t;
76   func_asserts: expr list;
77   func_ret_val: dim * expr;
78 }
79
80 type program = (variable StringMap.t) * (func_decl StringMap.t) * (extern_func
  StringMap.t)
81
82 type listable = Inits of init list |
83   Vars of var list |
84   Stmts of stmt list |
85   RawFuncs of raw_func list |
86   Externs of extern_func list |
87   Libraries of library list |
88   Exprs of expr list |
89   Rows of (expr list) list |
90   Strings of string list |
91   Cases of case list |
92   Formulas of formula list
93
94 exception IllegalRangeLiteral of string
95 exception TransformedAway of string
96
97 let quote_string str =
98   let escape_characters = Str.regexp "[\n \t \r \\ \"]" in
99   let replace_fn s = match Str.matched_string s with
100     "\n" -> "\\n" |
101     "\t" -> "\\t" |
102     "\r" -> "\\r" |
103     "\\\" -> "\\\\" |
104     "\"" -> "\\\"" |
105     _ -> Str.matched_string s in
106   "\"" ^ Str.global_substitute escape_characters replace_fn str ^ "\""
107
108 let string_of_op o = "\"" ^ (match o with
109   Plus -> "+" | Minus -> "-" | Times -> "*" | Divide -> "/" | Mod -> "%" | Pow ->
    "**" |
110   LShift -> "<<" | RShift -> ">>" | BitOr -> "|" | BitAnd -> "&" | BitXor -> "^" |
111   Eq -> "==" | Gt -> ">" | GtEq -> ">=" | Lt -> "<" | LtEq -> "<=" |
112   LogAnd -> "&&" | LogOr -> "||" ) ^ "\""
113
114 let string_of_unop = function

```

```

115     Neg -> "\"-\"" | LogNot -> "\"!\"" | BitNot -> "\"~\"" | Truthy -> "\"truthy\"" |
116     SizeOf -> "\"size\"" | TypeOf -> "\"type\"" | Row -> "\"row\"" | Column -> "\"
        column\""
117
118 let rec string_of_expr = function
119     LitInt(l) ->         "{\"LitInt\": \" ^ string_of_int l ^ \"}"
120   | LitFlt(l) ->         "{\"LitFlt\": \" ^ string_of_float l ^ \"}"
121   | LitString(s) ->       "{\"LitString\": \" ^ quote_string s ^ \"}"
122   | LitRange(rowlist) ->  "{\"LitRange\": \" ^ string_of_list (Rows rowlist) ^ \"}"
123   | Id(s) ->             "{\"Id\": \" ^ quote_string s ^ \"}"
124   | Empty ->             "\"Empty\""
125   | BinOp(e1, o, e2) ->   "{\"BinOp\": { \" ^
126                           "\"expr1\": \" ^ string_of_expr e1 ^ \", \" ^
127                           "\"operator\": \" ^ string_of_op o ^ \", \" ^
128                           "\"expr2\": \" ^ string_of_expr e2 ^ \" } }"
129   | UnOp(o, e) ->         "{\"UnOp\": { \" ^
130                           "\"operator\": \" ^ string_of_unop o ^ \", \" ^
131                           "\"expr\": \" ^ string_of_expr e ^ \" } }"
132   | Ternary(c, e1, e2) ->  "{\"Ternary\": { \" ^
133                           "\"condition\": \" ^ string_of_expr c ^ \", \" ^
134                           "\"ifExpr\": \" ^ string_of_expr e1 ^ \", \" ^
135                           "\"elseExpr\": \" ^ string_of_expr e2 ^ \" } }"
136   | ReducedTernary(s1, s2, s3) -> "{\"ReducedTernary\": { \" ^
137                                   "\"truthiness\": \" ^ quote_string s1 ^ \", \" ^
138                                   "\"true_values\": \" ^ quote_string s2 ^ \", \" ^
139                                   "\"false_values\": \" ^ quote_string s3 ^ \" } }"
140   | Switch(eo, cases, dflt) -> "{\"Switch\": { \" ^
141                                   "\"condition\": \" ^
142                                   (match eo with None -> "null" | Some e ->
143                                   string_of_expr e) ^ \", \" ^
143                                   "\"cases\": \" ^ string_of_list (Cases cases) ^ \", \" ^
144                                   "\"defaultExpr\": \" ^ string_of_expr dflt ^ \" } }"
145   | Call(f, arguments) ->  "{\"Call\": { \" ^
146                                   "\"function\": \" ^ quote_string f ^ \", \" ^
147                                   "\"arguments\": \" ^ string_of_list (Exprs arguments) ^
148                                   \" } }"
148   | Selection(e, s) ->     "{\"Selection\": { \" ^
149                                   "\"expr\": \" ^ string_of_expr e ^ \", \" ^
150                                   "\"slices\": \" ^ string_of_sel s ^ \" } }"
151   | Precedence(e1, e2) ->  "{\"Precedence\": { \" ^
152                                   "\"prior_expr\": \" ^ string_of_expr e1 ^ \", \" ^
153                                   "\"dependent_expr\": \" ^ string_of_expr e2 ^ \" } }"
154
155 and string_of_case (e1, e) =
156     "{\"Cases\": \" ^ string_of_list (Exprs e1) ^ \", \" ^
157     \"expr\": \" ^ string_of_expr e ^ \"}"
158
159 and string_of_sel (s1, s2) =
160     "{\"slice1\": \" ^ string_of_slice s1 ^ \", \"slice2\": \" ^ string_of_slice s2 ^ \"}"
161
162 and string_of_slice = function
163     None -> "null"
164   | Some (start_idx, end_idx) -> "{\"start\": \" ^ string_of_index start_idx ^ \", \"end
165                                   \": \" ^ string_of_index end_idx ^ \"}"
166
167 and string_of_index = function

```



```

167     None -> "null"
168   | Some(Abs(e)) -> "{\"Absolute\": " ^ string_of_expr e ^ "}"
169   | Some(Rel(e)) -> "{\"Relative\": " ^ string_of_expr e ^ "}"
170   | Some(DimensionStart) -> "\"DimensionStart\""
171   | Some(DimensionEnd) -> "\"DimensionEnd\""
172
173 and string_of_dim (d1,d2) = "{\"d1\": " ^ (match d1 with None -> "null" | Some e ->
    string_of_expr e) ^ ", " ^
174     "\"d2\": " ^ (match d2 with None -> "null" | Some e ->
    string_of_expr e) ^ "}"
175
176 and string_of_var (d, s) = "{\"Dimensions\": " ^ string_of_dim d ^ ", " ^
    "\"VarName\": " ^ quote_string s ^ "}"
177
178
179 and string_of_assign (s, selection, eo) =
180     "{\"VarName\": " ^ quote_string s ^ ", " ^
181     "\"Selection\": " ^ string_of_sel selection ^ ", " ^
182     "\"expr\": " ^ (match eo with None -> "null" | Some e -> string_of_expr e) ^ "}"
183
184 and string_of_varinit (d, inits) =
185     "{\"Dimensions\": " ^ string_of_dim d ^
186     ",\"Initializations\": " ^ string_of_list (Inits inits) ^ "}"
187
188 and string_of_init (s, eo) =
189     "{\"VarName\": " ^ quote_string s ^ ", " ^
190     "\"expr\": " ^ (match eo with None -> "null" | Some e -> string_of_expr e) ^ "}"
191
192 and string_of_stmt = function
193     Assign(a) -> "{\"Assign\": " ^ string_of_assign a ^ "}"
194   | Varinit(d, inits) -> "{\"Varinit\": " ^ string_of_varinit (d, inits) ^ "}"
195
196 and string_of_range (d, e) = "{\"Dimensions\": " ^ string_of_dim d ^ ", " ^
    "\"expr\": " ^ string_of_expr e ^ "}"
197
198
199 and string_of_raw_func fd =
200     "{\"Name\": " ^ quote_string fd.name ^ ", " ^
201     "\"Params\": " ^ string_of_list (Vars fd.params) ^ ", " ^
202     "\"Stmts\": " ^ string_of_list (Stmts fd.body) ^ ", " ^
203     "\"Assertions\": " ^ string_of_list (Exprs fd.raw_asserts) ^ ", " ^
204     "\"ReturnVal\": " ^ string_of_range fd.ret_val ^ "}"
205
206 and string_of_extern_func fd =
207     "{\"Name\": " ^ quote_string fd.extern_fn_name ^ ", " ^
208     "\"Params\": " ^ string_of_list (Vars fd.extern_fn_params) ^ ", " ^
209     "\"Library\": " ^ quote_string fd.extern_fn_libname ^ ", " ^
210     "\"ReturnDim\": " ^ string_of_dim fd.extern_ret_val ^ "}"
211
212 and string_of_library (Library(lib_name, lib_fns)) =
213     "{\"LibraryName\": " ^ quote_string lib_name ^ ", " ^
214     "\"ExternalFunctions\": " ^ string_of_list (Externs lib_fns) ^ "}"
215
216 and string_of_dimexpr = function
217     DimOneByOne -> "1"
218   | DimId(s) -> quote_string s
219
220 and string_of_formula f =

```

```

221   "{ \"RowStart\": \" ^ string_of_index (Some f.formula_row_start) ^ \", \" ^
222   \"RowEnd\": \" ^ string_of_index (f.formula_row_end) ^ \", \" ^
223   \"ColumnStart\": \" ^ string_of_index (Some f.formula_col_start) ^ \", \" ^
224   \"ColumnEnd\": \" ^ string_of_index (f.formula_col_end) ^ \", \" ^
225   \"Formula\": \" ^ string_of_expr f.formula_expr ^ \" }"
226
227   and string_of_list l =
228     let stringrep = (match l with
229       | Inits (il) -> List.map string_of_init il
230       | Vars (vl) -> List.map string_of_var vl
231       | Stmts (sl) -> List.map string_of_stmt sl
232       | RawFuncs (fl) -> List.map string_of_raw_func fl
233       | Externs (efl) -> List.map string_of_extern_func efl
234       | Libraries (libl) -> List.map string_of_library libl
235       | Exprs (el) -> List.map string_of_expr el
236       | Rows (rl) -> List.map (fun (el : expr list) -> string_of_list (Exprs el)) rl
237       | Strings (sl) -> List.map quote_string sl
238       | Cases (cl) -> List.map string_of_case cl
239       | Formulas (fl) -> List.map string_of_formula fl)
240     in "[" ^ String.concat ", " stringrep ^ "]"
241
242   let string_of_raw_program (imp, glb, fs, exts) =
243     "{ \"Program\": { \" ^
244       \"Imports\": \" ^ string_of_list (Strings imp) ^ \", \" ^
245       \"Globals\": \" ^ string_of_list (Stmts glb) ^ \", \" ^
246       \"ExternalLibraries\": \" ^ string_of_list (Libraries exts) ^ \", \" ^
247       \"Functions\": \" ^ string_of_list (RawFuncs fs) ^ \" } }"
248
249   let string_of_variable v =
250     "{ \"Rows\": \" ^ string_of_dimexpr v.var_rows ^ \", \" ^
251     \"Columns\": \" ^ string_of_dimexpr v.var_cols ^ \", \" ^
252     \"Formulas\": \" ^ string_of_list (Formulas v.var_formulas) ^ \" }"
253
254   let string_of_map value_desc val_printing_fn m =
255     let f_key_val_list k v l = (
256       "{ \"\" ^ value_desc ^ \"Name\": \" ^ quote_string k ^ \", \" ^
257       \"\" ^ value_desc ^ \"Def\": \" ^ val_printing_fn v ^ \" }"
258     ) :: l in
259     "[" ^ String.concat ", " (List.rev (StringMap.fold f_key_val_list m [])) ^ "]"
260
261   let string_of_funcdecl f =
262     "{ \"Params\": \" ^ string_of_list (Vars f.func_params) ^ \", \" ^
263     \"Variables\": \" ^ string_of_map \"Variable\" string_of_variable f.func_body ^ \", \" ^
264     \"Assertions\": \" ^ string_of_list (Exprs f.func_asserts) ^ \", \" ^
265     \"ReturnVal\": \" ^ string_of_range f.func_ret_val ^ \" }"
266
267   let string_of_program (glb, fs, exts) =
268     "{ \"Program\": { \" ^
269       \"Globals\": \" ^ string_of_map \"Variable\" string_of_variable glb ^ \", \" ^
270       \"Functions\": \" ^ string_of_map \"Function\" string_of_funcdecl fs ^ \", \" ^
271       \"ExternalFunctions\": \" ^ string_of_map \"ExternalFunctions\"
272         string_of_extern_func exts ^ \" } }"
273
274   let allow_range_literal = function
275     LitRange(rowlist) ->
276       let rec check_range_literal rl =

```

```

276     List.for_all (fun exprs -> List.for_all check_basic_expr exprs) rl
277     and check_basic_expr = function
278         LitInt(_) | UnOp(Neg, LitInt(_)) | LitFlt(_) | UnOp(Neg, LitFlt(_)) |
            LitString(_) | Empty -> true
279         | LitRange(rl) -> check_range_literal rl
280         | _ -> false in
281
282     if check_range_literal rowlist then LitRange(rowlist)
283     else raise(IllegalRangeLiteral(string_of_expr (LitRange(rowlist))))
284 | e -> raise(IllegalRangeLiteral(string_of_expr e))

```

## 9.4 transform.ml

```

1  (* jss2272 *)
2
3  open Ast
4  open Lexing
5  open Parsing
6  open Semant
7
8  module StringSet = Set.Make (String);;
9  let importSet = StringSet.empty;;
10
11  let idgen =
12      (* from http://stackoverflow.com/questions/10459363/side-effects-and-top-level-
        expressions-in-ocaml*)
13      let count = ref (-1) in
14      fun prefix -> incr count; "_tmp_" ^ prefix ^ string_of_int !count;;
15
16  let expand_file include_stdlib filename =
17      let print_error_location filename msg lexbuf =
18          let pos = lexbuf.lex_curr_p in
19          prerr_endline ("Syntax error in \"^ filename ^ "\": " ^ msg) ;
20          prerr_endline ("Line " ^ (string_of_int pos.pos_lnum) ^ " at character " ^ (
            string_of_int (pos.pos_cnum - pos.pos_bol))) in
21
22  let rec expand_imports processed_imports globals fns exts dir = function
23      [] -> ([], globals, fns, exts)
24      | (import, use_dir) :: imports ->
25          (* print_endline "-----";
26             print_endline ("Working on: " ^ import) ;
27             print_endline ("Already processed:"); *)
28          (* StringSet.iter (fun a -> print_endline a) processed_imports; *)
29          let in_chan = open_in import in
30          let lexbuf = (Lexing.from_channel (in_chan)) in
31          let (file_imports, file_globals, file_functions, file_extens) =
32              try Parser.program Scanner.token lexbuf
33              with
34                  Parsing.Parse_error -> print_error_location import "" lexbuf ; exit(-1)
35                  | Scanner.SyntaxError(s) -> print_error_location import s lexbuf ; exit(-1)
36          in
37          let file_imports = List.map (fun file -> (if use_dir then (dir ^ "/" ) else "" ) ^
            file) file_imports in
38          let new_proc = StringSet.add import processed_imports and _ = close_in in_chan
            in

```

```

39      (* print_endline ("Now I'm done with: ") ; *)
40      (* StringSet.iter (fun a -> print_endline a) new_proc; *)
41      let first_im_hearing_about imp = not (StringSet.mem imp new_proc || List.mem imp
      (List.map fst imports)) in
42      let new_imports = List.map (fun e -> (e, true)) (StringSet.elements (StringSet.
      of_list (List.filter first_im_hearing_about file_imports))) in
43      (* print_endline ("First I'm hearing about:") ; *)
44      (* List.iter print_endline new_imports; *)
45      expand_imports new_proc (globals @ file_globals) (fns @ file_functions) (exts @
      file_extens) (Filename.dirname import) (imports @ new_imports) in
46  expand_imports
47    StringSet.empty [] [] []
48    (Filename.dirname filename)
49    (if include_stdlib then [(filename, true); ("src/stdlib/stdlib.xtnd", false)] else
      [(filename, true)])
50
51  let expand_expressions (imports, globals, functions, externs) =
52    let lit_zero = LitInt(0) in let abs_zero = Abs(lit_zero) in
53    let lit_one = LitInt(1) in let abs_one = Abs(lit_one) in
54    let one_by_one = (Some lit_one, Some lit_one) in
55    let zero_comma_zero = (Some (Some abs_zero, Some abs_one),
56      Some (Some abs_zero, Some abs_one)) in
57    let entire_dimension = (Some DimensionStart, Some DimensionEnd) in
58    let entire_range = (Some entire_dimension, Some entire_dimension) in
59
60    let expand_expr expr_loc = function
61      (* Create a new variable for all expressions on the LHS to hold the result;
62      return the new expression and whatever new statements are necessary to create
63      the new variable *)
64      Empty -> raise (IllegalExpression("Empty not allowed in " ^ expr_loc))
65      | LitString(s) -> raise (IllegalExpression("String literal " ^ quote_string s ^ "
        not allowed in " ^ expr_loc))
66      | LitRange(rl) -> raise (IllegalExpression("Range literal " ^ string_of_list (Rows
        rl) ^ " not allowed in " ^ expr_loc))
67      | e -> let new_id = idgen expr_loc in (
68        Id(new_id),
69        [Varinit (one_by_one, [(new_id, None)]);
70        Assign (new_id, zero_comma_zero, Some e)]) in
71
72    let expand_index index_loc = function
73      (* Expand one index of a slice if necessary. *)
74      Abs(e) -> let (new_e, new_stmts) = expand_expr index_loc e in
75      (Abs(new_e), new_stmts)
76      | DimensionStart -> (DimensionStart, [])
77      | DimensionEnd -> (DimensionEnd, [])
78      | Rel(_) -> raise (IllegalExpression("relative - this shouldn't be possible")) in
79
80    let expand_slice slice_loc = function
81      (* Expand one or both sides as necessary. *)
82      None -> (entire_dimension, [])
83      | Some (Some (Abs(e)), None) ->
84      let (start_e, start_stmts) = expand_expr (slice_loc ^ "_start") e in
85      ((Some (Abs(start_e)), None), start_stmts)
86      | Some (Some idx_start, Some idx_end) ->
87      let (new_start, new_start_exprs) = expand_index (slice_loc ^ "_start") idx_start
      in

```

```

87     let (new_end, new_end_exprs) = expand_index (slice_loc ^ "_end") idx_end in
88     ((Some new_start, Some new_end), new_start_exprs @ new_end_exprs)
89   | Some (Some _, None) | Some (None, _) -> raise (IllegalExpression("Illegal slice
    - this shouldn't be possible")) in
90
91 let expand_assign asgn_loc (var_name, (row_slice, col_slice), formula) =
92   (* expand_assign: Take an Assign and return a list of more
93     atomic statements, with new variables replacing any
94     complex expressions in the selection slices and with single
95     index values desugared to expr:expr+1. *)
96   try
97     let (new_row_slice, row_exprs) = expand_slice (asgn_loc ^ "_" ^ var_name ^ "_row"
98       ^ row_slice) in
99     let (new_col_slice, col_exprs) = expand_slice (asgn_loc ^ "_" ^ var_name ^ "_col"
100       ^ col_slice) in
101     Assign(var_name, (Some new_row_slice, Some new_col_slice), formula) :: (
102       row_exprs @ col_exprs)
103   with IllegalExpression(s) ->
104     raise (IllegalExpression("Illegal expression (" ^ s ^ ") in " ^
105       string_of_assign (var_name, (row_slice, col_slice),
106         formula))) in
107
108 let expand_init (r, c) (v, e) =
109   Varinit((Some r, Some c), [(v, None)]) ::
110   match e with
111   | None -> []
112   | Some e -> [Assign (v, entire_range, Some e)] in
113
114 let expand_dimension dim_loc = function
115   | None -> expand_expr dim_loc (LitInt(1))
116   | Some e -> expand_expr dim_loc e in
117
118 let expand_varinit fname ((row_dim, col_dim), inits) =
119   (* expand_varinit: Take a Varinit and return a list of more atomic
120     statements. Each dimension will be given a temporary ID, which
121     will be declared as [1,1] _tmpXXX; the formula for tmpXXX will be
122     set as a separate assignment; the original variable will be
123     declared as [_tmpXXX, _tmpYYY] var; and the formula assignment
124     will be applied to [:,:]. *)
125   try
126     let (row_e, row_stmts) = expand_dimension (fname ^ "_" ^ (String.concat "_" (
127       List.map fst inits)) ^ "_row_dim") row_dim in
128     let (col_e, col_stmts) = expand_dimension (fname ^ "_" ^ (String.concat "_" (
129       List.map fst inits)) ^ "_col_dim") col_dim in
130     row_stmts @ col_stmts @ List.concat (List.map (expand_init (row_e, col_e)) inits
131       )
132   with IllegalExpression(s) ->
133     raise (IllegalExpression("Illegal expression (" ^ s ^ ") in " ^
134       string_of_varinit ((row_dim, col_dim), inits))) in
135
136 let expand_stmt fname = function
137   | Assign(a) -> expand_assign fname a
138   | Varinit(d, inits) -> expand_varinit fname (d, inits) in
139
140 let expand_stmt_list fname stmts = List.concat (List.map (expand_stmt fname) stmts)
141   in

```

```

134
135 let expand_params fname params =
136   let needs_sizevar = function
137     ((None, None), _) -> false
138     | _ -> true in
139   let params_with_sizevar = List.map (fun x -> (idgen (fname ^ "_" ^ (snd x) ^ "
    _size"), x)) (List.filter needs_sizevar params) in
140   let expanded_args = List.map (fun (sv, ((rv, cv), s)) -> ((sv, s), [((sv, abs_zero
    ), rv); ((sv, abs_one), cv)])) params_with_sizevar in
141   let (sizes, inits) = (List.map fst expanded_args, List.concat (List.map snd
    expanded_args)) in
142   let add_item (varset, (assertlist, initlist)) ((sizevar, pos), var) =
143     (match var with
144       Some Id(s) ->
145         if StringSet.mem s varset then
146           (* We've seen this variable before; don't initialize it, just assert it *)
147           (varset, (BinOp(Id(s), Eq, Selection(Id(sizevar), (Some(Some(pos), None),
    None))) :: assertlist, initlist))
148         else
149           (* We're seeing a string for the first time; don't assert it, just create
    it *)
150           (StringSet.add s varset, (assertlist,
151             Assign(s, zero_comma_zero, Some (Selection(Id(
    sizevar), (Some(Some(pos), None), None))) ::
152             Varinit(one_by_one, [(s, None)]) ::
153             initlist))
154       | Some LitInt(i) -> (* Seeing a number; don't do anything besides create an
    assertion *)
155       (varset, (BinOp(LitInt(i), Eq, Selection(Id(sizevar), (Some(Some(pos), None),
    None))) :: assertlist, initlist))
156       | Some e -> raise (IllegalExpression("Illegal expression (" ^ string_of_expr e
    ^ ") in function signature"))
157       | _ -> raise (IllegalExpression("Cannot supply a single dimension in function
    signature"))) in
158   let (rev_assertions, rev_inits) = snd (List.fold_left add_item (StringSet.empty,
    ([], [])) inits) in
159   let create_sizevar (sizevar, arg) = [
160     Varinit(one_by_one, [(sizevar, None)]);
161     Assign(sizevar, entire_range, Some(UnOp(SizeOf, Id(arg))))] in
162   (List.concat (List.map create_sizevar sizes), List.rev rev_assertions, List.rev
    rev_inits) in
163
164 let expand_function f =
165   let (new_sizevars, assertions, size_inits) = expand_params f.name f.params in
166   let new_retval_id = idgen (f.name ^ "_retval") in
167   let new_retval = Id(new_retval_id) in
168   let retval_inits = [Varinit (one_by_one, [(new_retval_id, None)]];
169     Assign (new_retval_id, zero_comma_zero, Some (snd f.ret_val))]
    in
170   let new_assert_id = idgen (f.name ^ "_assert") in
171   let add_assert al a = BinOp(al, LogAnd, a) in
172   let new_assert_expr = List.fold_left add_assert (LitInt(1)) assertions in
173   let new_assert = Id(new_assert_id) in
174   let assert_inits = [Varinit (one_by_one, [(new_assert_id, None)]];
175     Assign (new_assert_id, zero_comma_zero, Some new_assert_expr)]
    in

```

```

176 {
177   name = f.name;
178   params = f.params;
179   raw_asserts = [new_assert];
180   body = new_sizevars @ size_inits @ retval_inits @ assert_inits @
        expand_stmt_list f.name f.body;
181   ret_val = (fst f.ret_val, new_retval)
182 } in
183 (imports, expand_stmt_list "global" globals, List.map expand_function functions,
    externs);;

184
185 let create_maps (imports, globals, functions, externs) =
186   let vd_of_vi = function
187     (* vd_of_vi— Take a bare Varinit from the previous transformations
188       and return a (string, variable) pair *)
189     Varinit((Some r, Some c), [(v, None)]) -> (v, {
190       var_rows = (match r with
191         LitInt(1) -> DimOneByOne
192         | Id(s) -> DimId(s)
193         | _ -> raise (LogicError("Unrecognized expression for rows of " ^ v)));
194       var_cols = (match c with
195         LitInt(1) -> DimOneByOne
196         | Id(s) -> DimId(s)
197         | _ -> raise (LogicError("Unrecognized expression for rows of " ^ v)));
198       var_formulas = [];
199     })
200   | _ -> raise (LogicError("Unrecognized format for post-desugaring Varinit")) in
201
202   let add_formula m = function
203     Varinit(_,_) -> m
204   | Assign(var_name, (Some (Some row_start, row_end), Some (Some col_start, col_end
205     )), Some e) ->
206     if StringMap.mem var_name m
207     then (let v = StringMap.find var_name m in
208           StringMap.add var_name {v with var_formulas = v.var_formulas @ [{
209             formula_row_start = row_start;
210             formula_row_end = row_end;
211             formula_col_start = col_start;
212             formula_col_end = col_end;
213             formula_expr = e;
214           }]} m)
215     else raise (UnknownVariable(string_of_stmt (Assign(var_name, (Some (Some
216       row_start, row_end), Some (Some col_start, col_end)), Some e))))
217   | Assign(a) -> raise (LogicError("Unrecognized format for post-desugaring Assign:
218     " ^ string_of_stmt (Assign(a)))) in
219
220   let vds_of_stmts stmts =
221     let is_varinit = function Varinit(_,_) -> true | _ -> false in
222     let varinits = List.filter is_varinit stmts in
223     let vars_just_the_names = map_of_list (List.map vd_of_vi varinits) in
224     List.fold_left add_formula vars_just_the_names stmts in
225
226   let fd_of_raw_func f = (f.name, {
227     func_params = f.params;
228     func_body = vds_of_stmts f.body;
229     func_ret_val = f.ret_val;

```

```

227     func_asserts = f.raw_asserts;
228   }) in
229
230   let tupleize_library (Library(lib_name, lib_fns)) =
231     List.map (fun ext_fn -> (ext_fn.extern_fn_name, {ext_fn with extern_fn_libname =
232       lib_name})) lib_fns in
233
234   (vds_of_stmts globals,
235    map_of_list (List.map fd_of_raw_func functions),
236    map_of_list (List.concat (List.map tupleize_library externs)))
237
238   let single_formula e = {
239     formula_row_start = DimensionStart;
240     formula_row_end = Some DimensionEnd;
241     formula_col_start = DimensionStart;
242     formula_col_end = Some DimensionEnd;
243     formula_expr = e;
244   }
245
246   let ternarize_exprs (globals, functions, externs) =
247     let rec ternarize_expr lhs_var = function
248       BinOp(e1, LogAnd, e2) ->
249         let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
250         let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
251         (Ternary(UnOp(Truthy,new_e1), UnOp(Truthy,new_e2), LitInt(0)), new_e1_vars @
252           new_e2_vars)
253       | BinOp(e1, LogOr, e2) ->
254         let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
255         let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
256         (Ternary(UnOp(Truthy,new_e1), LitInt(1), UnOp(Truthy,new_e2)), new_e1_vars @
257           new_e2_vars)
258       | BinOp(e1, op, e2) ->
259         let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
260         let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
261         (BinOp(new_e1, op, new_e2), new_e1_vars @ new_e2_vars)
262       | UnOp(op, e) ->
263         let (new_e, new_e_vars) = ternarize_expr lhs_var e in
264         (UnOp(op, new_e), new_e_vars)
265       | Ternary(cond, e1, e2) ->
266         let (new_cond, new_cond_vars) = ternarize_expr lhs_var cond in
267         let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
268         let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
269         (Ternary(new_cond, new_e1, new_e2), new_cond_vars @ new_e1_vars @ new_e2_vars)
270       | Call(fname, args) ->
271         let new_args_and_vars = List.map (ternarize_expr lhs_var) args in
272         (Call(fname, (List.map fst new_args_and_vars)), List.concat (List.map snd
273           new_args_and_vars))
274       | Selection(e, (s11, s12)) ->
275         let (new_e, new_e_vars) = ternarize_expr lhs_var e in
276         let (new_s11, new_s11_vars) = ternarize_slice lhs_var s11 in
277         let (new_s12, new_s12_vars) = ternarize_slice lhs_var s12 in
278         (Selection(new_e, (new_s11, new_s12)), new_e_vars @ new_s11_vars @ new_s12_vars)
279       | Precedence(e1, e2) ->
280         let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
281         let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
282         (Precedence(new_e1, new_e2), new_e1_vars @ new_e2_vars)

```



```

279 | Switch(cond, cases, dflt) ->
280   ternarize_switch lhs_var cases dflt cond
281 (* | Debug(e) ->
282   let (new_e, new_e_vars) = ternarize_expr lhs_var e in
283   (Debug(new_e), new_e_vars) *)
284 | e -> (e, [])
285 and ternarize_switch lhs_var cases dflt cond =
286   let (new_cond_expr, new_cond_vars) = (match cond with
287     Some cond_expr ->
288       let (lhs_varname, lhs_vardef) = lhs_var in
289       let new_id = idgen (lhs_varname ^ "_switch_cond") in
290       let (new_e, new_e_vars) = ternarize_expr lhs_var cond_expr in
291       (Some (Selection (Id (new_id), (Some (Some (Rel (LitInt (0))), None), Some (Some (Rel (
292         LitInt (0))), None))), (new_id, {lhs_vardef with var_formulas = [single_formula new_e]})) ::
293       new_e_vars)
294   | None ->
295     (None, [])
296   ) in
297   let new_cases_and_vars = List.map (ternarize_case lhs_var new_cond_expr) cases in
298   let new_cases = List.map fst new_cases_and_vars in
299   let new_case_vars = List.concat (List.map snd new_cases_and_vars) in
300   let (new_dflt, new_dflt_vars) = ternarize_expr lhs_var dflt in
301   let rec combine_everything = function
302     [] -> new_dflt
303     | (combined_cases, e) :: more_cases -> Ternary(combined_cases, e,
304       combine_everything more_cases) in
305   and ternarize_case lhs_var cond (conds, e) =
306     let new_conds_and_vars = List.map (ternarize_expr lhs_var) conds in
307     let new_conds = List.map fst new_conds_and_vars in
308     let new_cond_vars = List.concat (List.map snd new_conds_and_vars) in
309     let (new_e, new_e_vars) = ternarize_expr lhs_var e in
310     let unify_case_cond_and_switch_cond case_cond = function
311       None -> case_cond
312       | Some switch_cond -> BinOp(switch_cond, Eq, case_cond) in
313     let rec unify_switch_cond_and_case_conds switch_cond = function
314       [case_cond] -> unify_case_cond_and_switch_cond case_cond switch_cond
315       | case_cond :: case_conds ->
316         let (combined_expr, _) = ternarize_expr lhs_var
317           (BinOp(unify_case_cond_and_switch_cond case_cond switch_cond, LogOr,
318             unify_switch_cond_and_case_conds switch_cond case_conds)) in
319         combined_expr
320     | [] -> raise (LogicError("Empty case condition list")) in
321   ((unify_switch_cond_and_case_conds cond new_conds, new_e), new_cond_vars @
322     new_e_vars)
323 and ternarize_slice lhs_var = function
324   None -> (None, [])
325   | Some (i1, i2) ->
326     let (new_i1, new_i1_vars) = ternarize_index lhs_var i1 in
327     let (new_i2, new_i2_vars) = ternarize_index lhs_var i2 in
328     (Some (new_i1, new_i2), new_i1_vars @ new_i2_vars)
329 and ternarize_index lhs_var = function
330   Some Abs(e) ->
331     let (new_e, new_e_vars) = ternarize_expr lhs_var e in
332     (Some (Abs(new_e)), new_e_vars)

```

```

331 | Some Rel(e) ->
332   let (new_e, new_e_vars) = ternarize_expr lhs_var e in
333   (Some(Rel(new_e)), new_e_vars)
334 | i -> (i, []) in
335 let ternarize_formula lhs_var f =
336   let (new_expr, new_vars) = ternarize_expr lhs_var f.formula_expr in
337   ({f with formula_expr = new_expr}, new_vars) in
338 let ternarize_variable varname vardef =
339   let new_formulas_and_vars = List.map (ternarize_formula (varname, vardef)) vardef.
340   var_formulas in
341   ({vardef with var_formulas = List.map fst new_formulas_and_vars}, List.concat (
342     List.map snd new_formulas_and_vars)) in
341 let ternarize_variables fn_name m =
342   let new_variables_and_maps = StringMap.mapi (fun varname vardef ->
343     ternarize_variable (fn_name ^ "_" ^ varname) vardef) m in
344   let add_item var_name (orig_var, new_vars) l = ((var_name, orig_var) :: fst l,
345     new_vars :: snd l) in
346   let combined_list = StringMap.fold add_item new_variables_and_maps ([],[]) in
347   map_of_list (List.rev (fst combined_list) @ List.concat (snd combined_list)) in
348 let ternarize_function fn_name fn_def = {fn_def with func_body = ternarize_variables
349   fn_name fn_def.func_body} in
350 (ternarize_variables "global" globals, StringMap.mapi ternarize_function functions,
351   externs)
352
353 let reduce_ternaries (globals, functions, externs) =
354   let rec reduce_expr lhs_var = function
355   | BinOp(e1, op, e2) ->
356     let (new_e1, new_e1_vars) = reduce_expr lhs_var e1 in
357     let (new_e2, new_e2_vars) = reduce_expr lhs_var e2 in
358     (BinOp(new_e1, op, new_e2), new_e1_vars @ new_e2_vars)
359   | UnOp(op, e) ->
360     let (new_e, new_e_vars) = reduce_expr lhs_var e in
361     (UnOp(op, new_e), new_e_vars)
362   | Ternary(cond, e1, e2) -> reduce_ternary lhs_var cond e1 e2
363   | Call(fname, args) ->
364     let new_args_and_vars = List.map (reduce_expr lhs_var) args in
365     (Call(fname, (List.map fst new_args_and_vars)), List.concat (List.map snd
366       new_args_and_vars))
367   | Selection(e, (sl1, sl2)) ->
368     let (new_e, new_e_vars) = reduce_expr lhs_var e in
369     let (new_sl1, new_sl1_vars) = reduce_slice lhs_var sl1 in
370     let (new_sl2, new_sl2_vars) = reduce_slice lhs_var sl2 in
371     (Selection(new_e, (new_sl1, new_sl2)), new_e_vars @ new_sl1_vars @ new_sl2_vars)
372   | Precedence(e1, e2) ->
373     let (new_e1, new_e1_vars) = reduce_expr lhs_var e1 in
374     let (new_e2, new_e2_vars) = reduce_expr lhs_var e2 in
375     (Precedence(new_e1, new_e2), new_e1_vars @ new_e2_vars)
376   (* | Debug(e) ->
377     let (new_e, new_e_vars) = reduce_expr lhs_var e in
378     (Debug(new_e), new_e_vars) *)
379   | e -> (e, [])
380 and reduce_ternary lhs_var cond e1 e2 =
381   let (new_cond, new_cond_vars) = reduce_expr lhs_var cond in
382   let (new_true_e, new_true_vars) = reduce_expr lhs_var e1 in
383   let (new_false_e, new_false_vars) = reduce_expr lhs_var e2 in
384   let (lhs_varname, lhs_vardef) = lhs_var in

```

```

380   let new_cond_id = idgen (lhs_varname ^ "_truthiness") in
381   let new_true_id = idgen (lhs_varname ^ "_values_if_true") in
382   let new_false_id = idgen (lhs_varname ^ "_values_if_false") in
383   (ReducedTernary(new_cond_id, new_true_id, new_false_id),
384    (new_cond_id, {lhs_vardef with var_formulas = [single_formula (UnOp(Truthy,
385      new_cond))]})) ::
386   (new_true_id, {lhs_vardef with var_formulas = [single_formula new_true_e]})) ::
387   (new_false_id, {lhs_vardef with var_formulas = [single_formula new_false_e]})) ::
388   (new_cond_vars @ new_true_vars @ new_false_vars))
389 and reduce_slice lhs_var = function
390   None -> (None, [])
391   | Some (i1, i2) ->
392     let (new_i1, new_i1_vars) = reduce_index lhs_var i1 in
393     let (new_i2, new_i2_vars) = reduce_index lhs_var i2 in
394     (Some (new_i1, new_i2), new_i1_vars @ new_i2_vars)
395 and reduce_index lhs_var = function
396   Some Abs(e) ->
397     let (new_e, new_e_vars) = reduce_expr lhs_var e in
398     (Some(Abs(new_e)), new_e_vars)
399   | Some Rel(e) ->
400     let (new_e, new_e_vars) = reduce_expr lhs_var e in
401     (Some(Rel(new_e)), new_e_vars)
402   | i -> (i, []) in
403 let reduce_formula lhs_var f =
404   let (new_expr, new_vars) = reduce_expr lhs_var f.formula_expr in
405   ({f with formula_expr = new_expr}, new_vars) in
406 let reduce_variable varname vardef =
407   let new_formulas_and_vars = List.map (reduce_formula (varname, vardef)) vardef.
408     var_formulas in
409   ({vardef with var_formulas = List.map fst new_formulas_and_vars}, List.concat (
410     List.map snd new_formulas_and_vars)) in
411 let reduce_variables fn_name m =
412   let new_variables_and_maps = StringMap.mapi (fun varname vardef -> reduce_variable
413     (fn_name ^ "_" ^ varname) vardef) m in
414   let add_item var_name (orig_var, new_vars) l = ((var_name, orig_var) :: fst l,
415     new_vars :: snd l) in
416   let combined_list = StringMap.fold add_item new_variables_and_maps ([],[]) in
417   map_of_list (List.rev (fst combined_list) @ List.concat (snd combined_list)) in
418 let reduce_function fn_name fn_def = {fn_def with func_body = reduce_variables
419   fn_name fn_def.func_body} in
420 (reduce_variables "global" globals, StringMap.mapi reduce_function functions,
421   externs)
422
423 let create_ast filename =
424   let ast_imp_res = expand_file true filename in
425   let ast_expanded = expand_expressions ast_imp_res in
426   let ast_mapped = create_maps ast_expanded in check_semantics ast_mapped ;
427   let ast_ternarized = ternarize_exprs ast_mapped in
428   let ast_reduced = reduce_ternaries ast_ternarized in check_semantics ast_reduced ;
429   ast_reduced

```

## 9.5 semant.ml

```

1 (* jss2272 *)
2

```

```

3 open Ast
4
5 exception IllegalExpression of string;;
6 exception DuplicateDefinition of string;;
7 exception UnknownVariable of string;;
8 exception UnknownFunction of string;;
9 exception WrongNumberArgs of string;;
10 exception LogicError of string;;
11
12 type symbol = LocalVariable of int | GlobalVariable of int | FunctionParameter of int
    | ExtendFunction of int
13 and symbolTable = symbol StringMap.t
14 and symbolTableType = Locals | Globals | ExtendFunctions
15
16 let map_of_list list_of_tuples =
17   (* map_of_list: Take a list of the form [("foo", 2); ("bar", 3)]
18     and create a StringMap using the first value of the tuple as
19     the key and the second value of the tuple as the value. Raises
20     an exception if the key appears more than once in the list. *)
21   let rec aux acc = function
22     [] -> acc
23     | t :: ts ->
24       if (StringMap.mem (fst t) acc) then raise(DuplicateDefinition(fst t))
25       else aux (StringMap.add (fst t) (snd t) acc) ts in
26   aux StringMap.empty list_of_tuples
27
28 let index_map table_type m =
29   let add_item key _ (accum_map, accum_idx) =
30     let index_val = match table_type with Locals -> LocalVariable(accum_idx) | Globals
31       -> GlobalVariable(accum_idx) | ExtendFunctions -> ExtendFunction(accum_idx) in
32     (StringMap.add key index_val accum_map, accum_idx + 1) in
33   StringMap.fold add_item m (StringMap.empty, 0)
34
35 let create_symbol_table global_symbols fn_def =
36   let (local_indices, _) = index_map Locals fn_def.func_body in
37   let add_param (st, idx) param_name =
38     let new_st = StringMap.add param_name (FunctionParameter(idx)) st in
39     (new_st, idx + 1) in
40   let (params_and_globals, _) = List.fold_left add_param (global_symbols, 0) (List.map
41     snd fn_def.func_params) in
42   StringMap.fold StringMap.add local_indices params_and_globals
43
44 let check_semantics (globals, functions, externs) =
45   let fn_signatures = map_of_list
46     ((StringMap.fold (fun s f l -> (s, List.length f.func_params) :: l) functions
47       []) @
48      (StringMap.fold (fun s f l -> (s, List.length f.extern_fn_params) :: l) externs
49        [])) in
50   let (global_symbols, _) = index_map Globals globals in
51
52   let check_call context called_fname num_args =
53     if (not (StringMap.mem called_fname fn_signatures)) then
54       (print_endline ("In " ^ context ^ "()", the undefined function " ^ called_fname ^
55         "()" was called");
56        raise(UnknownFunction(context ^ ", " ^ called_fname)))
57     else let signature_args = StringMap.find called_fname fn_signatures in

```

```

53     if num_args != signature_args then
54         (print_endline ("In " ^ context ^ "(), the function " ^ called_fname ^ "() was
                    called with " ^
55                     string_of_int num_args ^ " arguments " ^ "but the signature
                    specifies "
56                     ^ string_of_int signature_args) ;
57         raise(WrongNumberArgs(context ^ ", " ^ called_fname)))
58     else () in
59
60 let rec check_expr fname symbols = function
61     BinOp(e1,_,e2) -> check_expr fname symbols e1 ; check_expr fname symbols e2
62 | UnOp(_, e) -> check_expr fname symbols e
63 | Ternary(cond, e1, e2) -> check_expr fname symbols cond ; check_expr fname
    symbols e1 ; check_expr fname symbols e2
64 | ReducedTernary(s1, s2, s3) -> check_expr fname symbols (Id(s1)) ; check_expr
    fname symbols (Id(s2)) ; check_expr fname symbols (Id(s3))
65 | Id(s) -> if StringMap.mem s symbols then () else raise(UnknownVariable(fname ^
    "(): " ^ s))
66 | Switch(Some e, cases, dflt) -> check_expr fname symbols e ; List.iter (fun c ->
    check_case fname symbols c) cases ; check_expr fname symbols dflt
67 | Switch(None, cases, dflt) -> List.iter (fun c -> check_case fname symbols c)
    cases ; check_expr fname symbols dflt
68 | Call(called_fname, args) ->
69     check_call fname called_fname (List.length args) ;
70     List.iter (fun a -> check_expr fname symbols a) args
71 | Selection(e, (s11, s12)) -> check_expr fname symbols e ; check_slice fname
    symbols s11 ; check_slice fname symbols s12
72 | Precedence(e1, e2) -> check_expr fname symbols e1 ; check_expr fname symbols e2
73 (* | Debug(e) -> check_expr fname symbols e ; *)
74 | LitInt(_) | LitFlt(_) | LitRange(_) | LitString(_) | Empty -> ()
75 and check_case fname symbols (conds, e) = List.iter (fun c -> check_expr fname
    symbols c) conds ; check_expr fname symbols e
76 and check_slice fname symbols = function
77     None -> ()
78 | Some (i1, i2) -> check_index fname symbols i1 ; check_index fname symbols i2
79 and check_index fname symbols = function
80     Some Abs(e) -> check_expr fname symbols e
81 | Some Rel(e) -> check_expr fname symbols e
82 | _ -> () in
83 let check_formula fname symbols f =
84     check_index fname symbols (Some f.formula_row_start) ;
85     check_index fname symbols f.formula_row_end ;
86     check_index fname symbols (Some f.formula_col_start) ;
87     check_index fname symbols f.formula_col_end ;
88     check_expr fname symbols f.formula_expr in
89 let check_dim fname symbols = function
90     DimOneByOne -> ()
91 | DimId(s) -> check_expr fname symbols (Id(s)) in
92 let check_variable fname symbols v =
93     check_dim fname symbols v.var_rows ;
94     check_dim fname symbols v.var_cols ;
95     List.iter (fun f -> check_formula fname symbols f) v.var_formulas in
96 let check_variables context symbols vars =
97     StringMap.iter (fun _ v -> check_variable context symbols v) vars in
98
99 let check_function fname f =

```

```

100   if StringMap.mem fname externs then raise(DuplicateDefinition(fname ^ "() is
      defined as both an external and local function")) else ();
101   let locals = f.func_body in
102   let params = List.map snd f.func_params in
103   List.iter
104     (fun param ->
105       if StringMap.mem param locals then raise(DuplicateDefinition(param ^ " is
          defined multiple times in " ^ fname ^ "()))
106       else ())
107     params ;
108   let local_symbols = create_symbol_table global_symbols f in
109   check_variables fname local_symbols f.func_body ;
110   check_expr fname local_symbols (snd f.func_ret_val)
111
112   in check_variables "global_variables" global_symbols globals ; StringMap.iter
      check_function functions

```

## 9.6 codeGenTypes.ml

```

1  (*
2  jss2272
3  ns3158
4  *)
5
6  type something = {
7    var_instance_t : Llvm.lltype;
8    subrange_t : Llvm.lltype;
9    resolved_formula_t : Llvm.lltype;
10   value_t : Llvm.lltype;
11   dimensions_t : Llvm.lltype;
12   var_defn_t : Llvm.lltype;
13   var_defn_p : Llvm.lltype;
14   string_t : Llvm.lltype;
15   number_t : Llvm.lltype;
16   extend_scope_t : Llvm.lltype;
17   formula_t : Llvm.lltype;
18   formula_call_t : Llvm.lltype;
19   formula_p : Llvm.lltype;
20   formula_call_p : Llvm.lltype;
21   var_instance_p : Llvm.lltype;
22   subrange_p : Llvm.lltype;
23   resolved_formula_p : Llvm.lltype;
24   value_p : Llvm.lltype;
25   extend_scope_p : Llvm.lltype;
26   string_p : Llvm.lltype;
27   string_p_p : Llvm.lltype;
28   var_instance_p_p : Llvm.lltype;
29   int_t : Llvm.lltype;
30   long_t : Llvm.lltype;
31   flags_t : Llvm.lltype;
32   char_t : Llvm.lltype;
33   bool_t : Llvm.lltype;
34   void_t : Llvm.lltype;
35   char_p : Llvm.lltype;
36   char_p_p : Llvm.lltype;

```

```

37  (*void_p : Llvmltype;*)
38  float_t : Llvmltype;
39  rhs_index_t : Llvmltype;
40  rhs_slice_t : Llvmltype;
41  rhs_selection_t : Llvmltype;
42  rhs_index_p : Llvmltype;
43  rhs_slice_p : Llvmltype;
44  rhs_selection_p : Llvmltype;
45  };
46
47  type scope_field_type = VarDefn | VarInst | VarNum | ScopeRefCount | FunctionParams
48  let scope_field_type_index = function
49      VarDefn -> 0
50      | VarInst -> 1
51      | VarNum -> 2
52      | ScopeRefCount -> 3
53      | FunctionParams -> 4
54
55  type value_field_flags = Empty | Number | String | Range
56  let value_field_flags_index = function
57      Empty -> 0
58      | Number -> 1
59      | String -> 2
60      | Range -> 3
61  let int_to_type_array = ["Empty"; "Number"; "String"; "Range"]
62
63  type value_field = Flags | Number | String | Subrange
64  let value_field_index = function
65      Flags -> 0
66      | Number -> 1
67      | String -> 2
68      | Subrange -> 3
69
70  type var_defn_field = Rows | Cols | NumFormulas | Formulas | OneByOne | VarName
71  let var_defn_field_index = function
72      Rows -> 0
73      | Cols -> 1
74      | NumFormulas -> 2
75      | Formulas -> 3
76      | OneByOne -> 4
77      | VarName -> 5
78
79  type formula_field = FromFirstRow | RowStartNum | ToLastRow | RowEndNum |
      FromFirstCols | ColStartNum | ToLastCol | ColEndNum | IsSingleRow | IsSingleCol |
      FormulaCall
80  let formula_field_index = function
81      FromFirstRow -> 0
82      | RowStartNum -> 1
83      | ToLastRow -> 2
84      | RowEndNum -> 3
85      | FromFirstCols -> 4
86      | ColStartNum -> 5
87      | ToLastCol -> 6
88      | ColEndNum -> 7
89      | IsSingleRow -> 8
90      | IsSingleCol -> 9

```

```

91 | FormulaCall -> 10
92
93 type var_instance_field = Rows | Cols | NumFormulas | Formulas | Closure | Values |
    Status
94 let var_instance_field_index = function
95     Rows -> 0
96     | Cols -> 1
97     | NumFormulas -> 2
98     | Formulas -> 3
99     | Closure -> 4
100    | Values -> 5
101    | Status -> 6
102
103 type var_instance_status_flags = NeverExamined | Calculated | InProgress
104 let var_instance_status_flags_index = function
105     NeverExamined -> 0
106     | Calculated -> 2
107     | InProgress -> 4
108
109 type subrange_field = BaseRangePtr | BaseOffsetRow | BaseOffsetCol | SubrangeRows |
    SubrangeCols
110 let subrange_field_index = function
111     BaseRangePtr -> 0
112     | BaseOffsetRow -> 1
113     | BaseOffsetCol -> 2
114     | SubrangeRows -> 3
115     | SubrangeCols -> 4
116
117 type dimensions_field = DimensionRows | DimensionCols
118 let dimensions_field_index = function
119     DimensionRows -> 0
120     | DimensionCols -> 1
121
122 type string_field = StringCharPtr | StringLen | StringRefCount
123 let string_field_index = function
124     StringCharPtr -> 0
125     | StringLen -> 1
126     | StringRefCount -> 2
127
128 type rhs_index_field = RhsExprVal | RhsIndexType
129 let rhs_index_field_index = function
130     RhsExprVal -> 0
131     | RhsIndexType -> 1
132
133 type rhs_index_type_flags = RhsIdxAbs | RhsIdxRel | RhsIdxDimStart | RhsIdxDimEnd
134 let rhs_index_type_flags_const = function
135     RhsIdxAbs -> 0
136     | RhsIdxRel -> 1
137     | RhsIdxDimStart -> 2
138     | RhsIdxDimEnd -> 4 (* No 3 *)
139
140 type rhs_slice_field = RhsSliceStartIdx | RhsSliceEndIdx
141 let rhs_slice_field_index = function
142     RhsSliceStartIdx -> 0
143     | RhsSliceEndIdx -> 1
144

```



```

145 type rhs_selection_field = RhsSelSlice1 | RhsSelSlice2
146 let rhs_selection_field_index = function
147     RhsSelSlice1 -> 0
148     | RhsSelSlice2 -> 1
149
150 let setup_types ctx =
151     let var_instance_t = LlvM.named_struct_type ctx "var_instance" (*Range struct is a 2
152         D Matrix of values*)
153     and subrange_t = LlvM.named_struct_type ctx "subrange" (*Subrange is a wrapper
154         around a range to cut cells*)
155     and int_t = LlvM.i32_type ctx (*Integer*)
156     and long_t = LlvM.i64_type ctx
157     and float_t = LlvM.double_type ctx
158     and flags_t = LlvM.i8_type ctx (*Flags for statuses*)
159     and char_t = LlvM.i8_type ctx (*Simple ASCII character*)
160     and bool_t = LlvM.i1_type ctx (*boolean 0 = false, 1 = true*)
161     and void_t = LlvM.void_type ctx (**)
162     and value_t = LlvM.named_struct_type ctx "value" (*Value encapsulates the content of
163         a cell*)
164     and dimensions_t = LlvM.named_struct_type ctx "dimensions" (**)
165     and resolved_formula_t = LlvM.named_struct_type ctx "resolved_formula"
166     and extend_scope_t = LlvM.named_struct_type ctx "extend_scope"
167     and var_defn_t = LlvM.named_struct_type ctx "var_def"
168     and formula_t = LlvM.named_struct_type ctx "formula"
169     and string_t = LlvM.named_struct_type ctx "string" in
170     let var_instance_p = (LlvM.pointer_type var_instance_t)
171     and var_defn_p = LlvM.pointer_type var_defn_t
172     and resolved_formula_p = (LlvM.pointer_type resolved_formula_t)
173     and subrange_p = (LlvM.pointer_type subrange_t)
174     and value_p = (LlvM.pointer_type value_t)
175     and value_p_p = (LlvM.pointer_type (LlvM.pointer_type value_t))
176     and extend_scope_p = (LlvM.pointer_type extend_scope_t)
177     and char_p = (LlvM.pointer_type char_t)
178     and string_p = (LlvM.pointer_type string_t)
179     and char_p_p = (LlvM.pointer_type (LlvM.pointer_type char_t))
180     and string_p_p = (LlvM.pointer_type (LlvM.pointer_type string_t))
181     and number_t = float_t
182     and formula_p = (LlvM.pointer_type formula_t) in
183     let rhs_index_t = LlvM.named_struct_type ctx "rhs_index"
184     and rhs_slice_t = LlvM.named_struct_type ctx "rhs_slice"
185     and rhs_selection_t = LlvM.named_struct_type ctx "rhs_selection" in
186     let rhs_index_p = LlvM.pointer_type rhs_index_t
187     and rhs_slice_p = LlvM.pointer_type rhs_slice_t
188     and rhs_selection_p = LlvM.pointer_type rhs_selection_t
189     (*and void_p = (LlvM.pointer_type void_t)*) in
190     let var_instance_p_p = (LlvM.pointer_type var_instance_p)
191     and formula_call_t = (LlvM.function_type value_p [|extend_scope_p(*scope*); int_t(*
192         row*); int_t(*col*)|]) in
193     let formula_call_p = LlvM.pointer_type formula_call_t in
194     let _ = LlvM.struct_set_body rhs_index_t (Array.of_list [
195         value_p (*val_of_expr*);
196         char_t (*rhs_index_type*);
197     ]) false in
198     let _ = LlvM.struct_set_body rhs_slice_t (Array.of_list [
199         rhs_index_p (*slice start index*);
200         rhs_index_p (*slice end index*);

```

```

197     }) false in
198   let _ = LlvM.struct_set_body rhs_selection_t (Array.of_list [
199     rhs_slice_p (*first slice*);
200     rhs_slice_p (*second slice*);
201   ]) false in
202   let _ = LlvM.struct_set_body var_instance_t (Array.of_list [
203     int_t(*rows*);
204     int_t(*columns*);
205     int_t(*numFormulas*);
206     resolved_formula_p(*formula with resolved dimensions*);
207     extend_scope_p(*scope that contains all variables of a function*);
208     value_p_p(*2D array of cell values*);
209     char_p(*2D array of calculation status for each cell*);
210     char_p(*Name*);
211   ]) false
212   and _ = LlvM.struct_set_body var_defn_t (Array.of_list [
213     int_t(*Rows*);
214     int_t(*Cols*);
215     int_t(*Number of formulas*);
216     formula_p;
217     char_t(*Is one by one range*);
218     char_p(*Name*);
219   ]) false
220   and _ = LlvM.struct_set_body formula_t (Array.of_list [
221     char_t (*from First row*);
222     int_t (*row Start num*);
223     char_t (*to last row*);
224     int_t (*row end num*);
225     char_t (*from first col*);
226     int_t (*col start*);
227     char_t (*to last col*);
228     int_t (*col end num*);
229     char_t (* is single row *);
230     char_t (* is single col *);
231     formula_call_p (*formula to call*);
232   ]) false
233   and _ = LlvM.struct_set_body extend_scope_t (Array.of_list [
234     var_defn_p(*variable definitions*);
235     var_instance_p_p(*variable instances*);
236     int_t(*number of variables*);
237     int_t(*reference count*);
238     LlvM.pointer_type value_p;
239   ]) false
240   and _ = LlvM.struct_set_body subrange_t (Array.of_list [
241     var_instance_p(*The target range*);
242     int_t(*row offset*);
243     int_t(*column offset*);
244     int_t(*row count*);
245     int_t(*column count*)
246   ]) false
247   and _ = LlvM.struct_set_body value_t (Array.of_list [
248     flags_t (*First bit indicates whether it is an int or a range*);
249     number_t (*Numeric value of the cell*);
250     string_p (*String value of the cell if applicable*);
251     subrange_p (*Range value of the cell if applicable*);
252     (*float_t (Double value of the cell*)

```

```

253     }) false
254   and _ = LlvM.struct_set_body string_t (Array.of_list [
255       char_p (*Pointer to null-terminated string*);
256       long_t (*Length of string*);
257       int_t (*Reference count*)
258     ]) false
259   and _ = LlvM.struct_set_body dimensions_t (Array.of_list [int_t; int_t]) false in
260   {
261     var_instance_t = var_instance_t;
262     value_t = value_t;
263     subrange_t = subrange_t;
264     resolved_formula_t = resolved_formula_t;
265     dimensions_t = dimensions_t;
266     number_t = number_t;
267     string_t = string_t;
268     extend_scope_t = extend_scope_t;
269     formula_t = formula_t;
270     formula_call_t = formula_call_t;
271
272     var_defn_t = var_defn_t;
273     var_defn_p = var_defn_p;
274     var_instance_p = var_instance_p;
275     subrange_p = subrange_p;
276     value_p = value_p;
277     resolved_formula_p = resolved_formula_p;
278     string_p = string_p;
279     char_p = char_p;
280     extend_scope_p = extend_scope_p;
281     formula_p = formula_p;
282     formula_call_p = formula_call_p;
283
284     var_instance_p_p = var_instance_p_p;
285
286     int_t = int_t;
287     long_t = long_t;
288     float_t = float_t;
289     flags_t = flags_t;
290     bool_t = bool_t;
291     char_t = char_t;
292     void_t = void_t;
293     char_p_p = char_p_p;
294     string_p_p = string_p_p;
295
296     rhs_index_t = rhs_index_t;
297     rhs_slice_t = rhs_slice_t;
298     rhs_selection_t = rhs_selection_t;
299     rhs_index_p = rhs_index_p;
300     rhs_slice_p = rhs_slice_p;
301     rhs_selection_p = rhs_selection_p;
302   }

```

## 9.7 codegen.ml

```

1  (*
2  Extend code generator

```

```

3   jss2272
4   ns3158
5   *)
6
7   open Ast
8   open Semant
9   open CodeGenTypes
10  exception NotImplemented
11
12  let runtime_functions = Hashtbl.create 20
13
14  let (=>) struct_ptr elem = (fun val_name builder ->
15    let the_pointer = LlvM.build_struct_gep struct_ptr elem "the_pointer" builder in
16    LlvM.build_load the_pointer val_name builder);;
17
18  let ($>) val_to_store (struct_ptr, elem) = (fun builder ->
19    let the_pointer = LlvM.build_struct_gep struct_ptr elem "" builder in
20    LlvM.build_store val_to_store the_pointer builder);;
21
22  (* from http://stackoverflow.com/questions/243864/what-is-the-ocaml-idiom-equivalent-to-pythons-range-function-without-the-infix *)
23  let zero_until i =
24    let rec aux n acc =
25      if n < 0 then acc else aux (n-1) (n :: acc)
26    in aux (i-1) []
27
28  let create_runtime_functions ctx bt the_module =
29    let add_runtime_func fname returntype arglist =
30      let the_func = LlvM.declare_function fname (LlvM.function_type returntype arglist)
31      the_module
32    in Hashtbl.add runtime_functions fname the_func in
33    add_runtime_func "strlen" bt.long_t [|bt.char_p|];
34    add_runtime_func "strcmp" bt.long_t [|bt.char_p; bt.char_p|];
35    add_runtime_func "pow" bt.float_t [|bt.float_t; bt.float_t|];
36    add_runtime_func "lrint" bt.int_t [|bt.float_t|];
37    add_runtime_func "llvm.memcpy.p0i8.p0i8.i64" bt.void_t [|bt.char_p; bt.char_p; bt.
38      long_t; bt.int_t; bt.bool_t|];
39    add_runtime_func "incStack" bt.void_t [|]|;
40    add_runtime_func "getVal" bt.value_p [|bt.var_instance_p; bt.int_t; bt.int_t|];
41    add_runtime_func "rg_eq" bt.int_t [|bt.value_p; bt.value_p|];
42    add_runtime_func "clone_value" bt.value_p [|bt.value_p|];
43    (* add_runtime_func "freeMe" (LlvM.void_type ctx) [|bt.extend_scope_p|]; *)
44    add_runtime_func "getSize" bt.value_p [|bt.var_instance_p|];
45    add_runtime_func "get_variable" bt.var_instance_p [|bt.extend_scope_p; bt.int_t|];
46    add_runtime_func "null_init" (LlvM.void_type ctx) [|bt.extend_scope_p|];
47    add_runtime_func "debug_print" (LlvM.void_type ctx) [|bt.value_p ; bt.char_p|];
48    add_runtime_func "new_string" bt.value_p [|bt.char_p|];
49    add_runtime_func "deref_subrange_p" bt.value_p [|bt.subrange_p|];
50    add_runtime_func "debug_print_selection" (LlvM.void_type ctx) [|bt.rhs_selection_p
51      |];
52    add_runtime_func "extract_selection" bt.value_p [|bt.value_p; bt.rhs_selection_p; bt
53      .int_t; bt.int_t|];
54    add_runtime_func "box_command_line_args" bt.value_p [|bt.int_t; bt.char_p_p|];
55    add_runtime_func "verify_assert" (LlvM.void_type ctx) [|bt.value_p; bt.char_p|];
56    ()
57

```

```

54 let translate (globals, functions, externs) =
55
56   (* LLVM Boilerplate *)
57   let context = LlvM.global_context () in
58   let base_module = LlvM.create_module context "Extend" in
59   let base_types = setup_types context in
60
61   (* Declare the runtime functions that we need to call *)
62   create_runtime_functions context base_types base_module ;
63
64   (* Build function_llvalues, which is a StringMap from function name to llvalue.
65    * It includes both functions from external libraries, such as the standard library,
66    * and functions declared within Extend. *)
67   let declare_library_function fname func accum_map =
68     let llvm_ftype = LlvM.function_type base_types.value_p (Array.of_list (List.map (
69       fun a -> base_types.value_p) func.extern_fn_params))) in
70     let llvm_fname = "extend_" ^ fname in
71     let llvm_fn = LlvM.declare_function llvm_fname llvm_ftype base_module in
72     StringMap.add fname llvm_fn accum_map in
73   let library_functions = StringMap.fold declare_library_function externs StringMap.
74     empty in
75
76   let define_user_function fname func =
77     let llvm_fname = "extend_" ^ fname in
78     let llvm_ftype = LlvM.function_type base_types.value_p (Array.of_list (List.map (
79       fun a -> base_types.value_p) func.func_params))) in
80     let llvm_fn = LlvM.define_function llvm_fname llvm_ftype base_module in
81     (func, llvm_fn) in
82   let extend_functions = StringMap.mapi define_user_function functions in
83   let function_llvalues = StringMap.fold StringMap.add (StringMap.map snd
84     extend_functions) library_functions in
85
86   (* Build the global symbol table *)
87   let (global_symbols, num_globals) = index_map Globals globals in
88   let (extend_fn_numbers, num_extend_fns) = index_map ExtendFunctions extend_functions
89     in
90
91   (* Create the global array that will hold each function's array of var_defns. *)
92   let vardefn_ptr = LlvM.const_pointer_null base_types.var_defn_p in
93   let vardefn_array = Array.make (StringMap.cardinal extend_functions) vardefn_ptr in
94   let array_of_vardefn_ptrs = LlvM.define_global "array_of_vardefn_ptrs" (LlvM.
95     const_array base_types.var_defn_p vardefn_array) base_module in
96
97   (* Create the pointer to the global scope object *)
98   let global_scope_loc = LlvM.define_global "global_scope_loc" (LlvM.
99     const_pointer_null base_types.extend_scope_p) base_module in
100
101   let main_def = LlvM.define_function "main" (LlvM.function_type base_types.int_t [|
102     base_types.int_t; base_types.char_p_p|]) base_module in
103   let main_bod = LlvM.builder_at_end context (LlvM.entry_block main_def) in
104
105   let init_def = LlvM.define_function "initialize_vardefns" (LlvM.function_type (LlvM.
106     void_type context) [||]) base_module in
107   let init_bod = LlvM.builder_at_end context (LlvM.entry_block init_def) in
108
109   let literal_def = LlvM.define_function "initialize_literals" (LlvM.function_type (
110     LlvM.void_type context) [||]) base_module in

```

```

100 let literal_bod = LlvM.builder_at_end context (LlvM.entry_block literal_def) in
101
102 (* Create the array of value_ps that will contain the responses to TypeOf(val) *)
103 let null_val_ptr = LlvM.const_pointer_null base_types.value_p in
104 let null_val_array = Array.make (Array.length int_to_type_array) null_val_ptr in
105 let array_of_typeof_val_ptrs = LlvM.define_global "array_of_val_ptrs" (LlvM.
    const_array base_types.value_p null_val_array) base_module in
106 let create_typeof_string i s =
107     let sp = LlvM.build_global_stringptr s "global_typeof_stringptr" literal_bod in
108     let vp = LlvM.build_call (Hashtbl.find runtime_functions "new_string") [|sp|] "
        global_typeof_string" literal_bod in
109     let vp_dst = LlvM.build_in_bounds_gep array_of_typeof_val_ptrs [|LlvM.const_int
        base_types.int_t 0; LlvM.const_int base_types.int_t i|] ("global_typeof_dst")
        literal_bod in
110     let _ = LlvM.build_store vp vp_dst literal_bod in
111     () in
112 Array.iteri create_typeof_string int_to_type_array ;
113
114 (* Look these two up once and for all *)
115 (* let deepCopy = Hashtbl.find runtime_functions "deepCopy" in *)
116 (* let freeMe = Hashtbl.find runtime_functions "freeMe" in *)
117 let getVal = Hashtbl.find runtime_functions "getVal" in (*getVal retrieves the value
    of a variable instance for a specific x and y*)
118 let getVar = Hashtbl.find runtime_functions "get_variable" in (*getVar retrieves a
    variable instance based on the offset. It instantiates the variable if it does
    not exist yet*)
119
120 (* build_formula_function takes a symbol table and an expression, builds the LLVM
    function, and returns the llvalue of the function *)
121 let build_formula_function (varname, formula_idx) symbols formula_expr =
122     let form_decl = LlvM.define_function ("formula_fn_" ^ varname ^ "_num_" ^ (
        string_of_int formula_idx)) base_types.formula_call_t base_module in
123     let builder_at_top = LlvM.builder_at_end context (LlvM.entry_block form_decl) in
124     let local_scope = LlvM.param form_decl 0 in
125     let cell_row = LlvM.param form_decl 1 in
126     let cell_col = LlvM.param form_decl 2 in
127     let global_scope = LlvM.build_load global_scope_loc "global_scope" builder_at_top
        in
128
129     (* Some repeated stuff to avoid cut & paste *)
130     let empty_type = (LlvM.const_int base_types.char_t (value_field_flags_index Empty)
        ) in
131     let number_type = (LlvM.const_int base_types.char_t (value_field_flags_index
        Number)) in
132     let string_type = (LlvM.const_int base_types.char_t (value_field_flags_index
        String)) in
133     let range_type = (LlvM.const_int base_types.char_t (value_field_flags_index Range)
        ) in
134     let make_block blockname =
135         let new_block = LlvM.append_block context blockname form_decl in
136         let new_builder = LlvM.builder_at_end context new_block in
137         (new_block, new_builder) in
138     let store_number value_ptr store_builder number_llvalue =
139         let sp = LlvM.build_struct_gep value_ptr (value_field_index Number) "num_pointer
        " store_builder in
140         let _ = LlvM.build_store number_type (LlvM.build_struct_gep value_ptr (

```

```

    value_field_index Flags) "" store_builder) store_builder in
141   ignore (Llvm.build_store number_llvalue sp store_builder) in
142   let store_empty value_ptr store_builder =
143     ignore (Llvm.build_store empty_type (Llvm.build_struct_gep value_ptr (
      value_field_index Flags) "" store_builder) store_builder) in
144
145   let make_truthiness_blocks blockprefix ret_val =
146     let (merge_bb, merge_builder) = make_block (blockprefix ^ "_merge") in
147
148     let (make_true_bb, make_true_builder) = make_block (blockprefix ^ "_true") in
149     let _ = store_number ret_val make_true_builder (Llvm.const_float base_types.
      float_t 1.0) in
150     let _ = Llvm.build_br merge_bb make_true_builder in
151
152     let (make_false_bb, make_false_builder) = make_block (blockprefix ^ "_false") in
153     let _ = store_number ret_val make_false_builder (Llvm.const_float base_types.
      float_t 0.0) in
154     let _ = Llvm.build_br merge_bb make_false_builder in
155
156     let (make_empty_bb, make_empty_builder) = make_block (blockprefix ^ "_empty") in
157     let _ = store_empty ret_val make_empty_builder in
158     let _ = Llvm.build_br merge_bb make_empty_builder in
159
160     (make_true_bb, make_false_bb, make_empty_bb, merge_builder) in
161
162   let rec build_expr old_builder exp = match exp with
163     LitInt(i) -> let vvv = Llvm.const_float base_types.float_t (float_of_int i) in
164     let ret_val = Llvm.build_malloc base_types.value_t "int_ret_val" old_builder
      in
165     let _ = store_number ret_val old_builder vvv in
166     (ret_val, old_builder)
167   | LitFlt(f) -> let vvv = Llvm.const_float base_types.float_t f in
168     let ret_val = Llvm.build_malloc base_types.value_t "flt_ret_val" old_builder
      in
169     let _ = store_number ret_val old_builder vvv in
170     (ret_val, old_builder)
171   | UnOp(Neg, LitInt(i)) -> build_expr old_builder (LitInt(-i))
172   | UnOp(Neg, LitFlt(f)) -> build_expr old_builder (LitFlt(-f))
173   | Empty ->
174     let ret_val = Llvm.build_malloc base_types.value_t "empty_ret_val" old_builder
      in
175     let _ = store_empty ret_val old_builder in
176     (ret_val, old_builder)
177   (* | Debug(e) ->
178     let (ret_val, new_builder) = build_expr old_builder e in
179     let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
      ret_val; Llvm.const_pointer_null base_types.char_p|] "" new_builder in
180     (ret_val, new_builder) *)
181   | Id(name) ->
182     let create_and_deref_subrange appropriate_scope i =
183       let llvm_var = Llvm.build_call getVar [|appropriate_scope; Llvm.const_int
        base_types.int_t i|] "llvm_var" old_builder in
184       let base_var_num_rows = (llvm_var => (var_instance_field_index Rows)) "
        base_var_num_rows" old_builder in
185       let base_var_num_cols = (llvm_var => (var_instance_field_index Cols)) "
        base_var_num_rows" old_builder in

```

```

186     let subrange_ptr = LlvM.build_alloca base_types.subrange_t "subrange_ptr"
187     old_builder in
188     let _ = (llvm_var $> (subrange_ptr, (subrange_field_index BaseRangePtr)))
189     old_builder in
190     let _ = ((LlvM.const_null base_types.int_t) $> (subrange_ptr, (
191     subrange_field_index BaseOffsetRow))) old_builder in
192     let _ = ((LlvM.const_null base_types.int_t) $> (subrange_ptr, (
193     subrange_field_index BaseOffsetCol))) old_builder in
194     let _ = (base_var_num_rows $> (subrange_ptr, (subrange_field_index
195     SubrangeRows))) old_builder in
196     let _ = (base_var_num_cols $> (subrange_ptr, (subrange_field_index
197     SubrangeCols))) old_builder in
198     (LlvM.build_call (Hashtbl.find runtime_functions "deref_subrange_p") [|
199     subrange_ptr|] "local_id_ret_val" old_builder, old_builder) in
200     (
201     match (try StringMap.find name symbols with Not_found -> raise(LogicError("
202     Something went wrong with your semantic analysis - " ^ name ^ " not found
203     "))) with
204     | LocalVariable(i) -> create_and_deref_subrange local_scope i
205     | GlobalVariable(i) -> create_and_deref_subrange global_scope i
206     | FunctionParameter(i) ->
207     let paramarray = (local_scope => (scope_field_type_index FunctionParams))
208     "paramarray" old_builder in
209     let param_addr = LlvM.build_in_bounds_gep paramarray [|LlvM.const_int
210     base_types.int_t i|] "param_addr" old_builder in
211     let param = LlvM.build_load param_addr "param" old_builder in
212     (LlvM.build_call (Hashtbl.find runtime_functions "clone_value") [|param|]
213     "function_param_ret_val" old_builder, old_builder)
214     | ExtendFunction(i) -> raise(LogicError("Something went wrong with your
215     semantic analysis - function " ^ name ^ " used as variable in RHS for " ^
216     varname))
217     )
218     | ReducedTernary(cond_var, true_var, false_var) ->
219     let get_llvm_var name getvar_builder =
220     match (try StringMap.find name symbols with Not_found -> raise(LogicError("
221     Something went wrong with your transformation - Reduced Ternary name " ^
222     name ^ " not found"))) with
223     | LocalVariable(i) -> LlvM.build_call getVar [|local_scope; LlvM.const_int
224     base_types.int_t i|] "llvm_var" getvar_builder
225     | GlobalVariable(i) -> LlvM.build_call getVar [|global_scope; LlvM.const_int
226     base_types.int_t i|] "llvm_var" getvar_builder
227     | _ -> raise(LogicError("Something went wrong with your transformation -
228     Reduced Ternary name " ^ name ^ " not a local or global variable")) in
229     let (empty_bb, empty_builder) = make_block "empty" in
230     let (not_empty_bb, not_empty_builder) = make_block "not_empty" in
231     let (truthy_bb, truthy_builder) = make_block "truthy" in
232     let (falsey_bb, falsey_builder) = make_block "falsey" in
233     let (merge_bb, merge_builder) = make_block "merge" in
234     let ret_val_addr = LlvM.build_alloca base_types.value_p "tern_ret_val_addr"
235     old_builder in
236     let cond_llvm_var = get_llvm_var cond_var old_builder in
237     let cond_val = LlvM.build_call getVal [|cond_llvm_var; cell_row; cell_col|] "
238     cond_val" old_builder in
239     let cond_val_type = (cond_val => (value_field_index Flags)) "cond_val_type"

```



```

    old_builder in
221   let is_empty = LlvM.build_icmp LlvM.Icmp.Eq empty_type cond_val_type "is_empty
      " old_builder in
222   let _ = LlvM.build_cond_br is_empty empty_bb not_empty_bb old_builder in
223
224   (* Empty basic block: *)
225   let ret_val_empty = LlvM.build_malloc base_types.value_t "tern_empty"
      empty_builder in
226   let _ = store_empty ret_val_empty empty_builder in
227   let _ = LlvM.build_store ret_val_empty ret_val_addr empty_builder in
228   let _ = LlvM.build_br merge_bb empty_builder in
229
230   (* Not empty basic block: *)
231   let the_number = (cond_val => (value_field_index Number)) "the_number"
      not_empty_builder in
232   let is_not_zero = LlvM.build_fcmp LlvM.Fcmp.One the_number (LlvM.const_float
      base_types.number_t 0.0) "is_not_zero" not_empty_builder in (* Fcmp.One =
      Not equal *)
233   let _ = LlvM.build_cond_br is_not_zero truthy_bb falsey_bb not_empty_builder
      in
234
235   (* Truthy basic block: *)
236   let truthy_llvm_var = get_llvm_var true_var truthy_builder in
237   let truthy_val = LlvM.build_call getVal [|truthy_llvm_var; cell_row; cell_col
      |] "truthy_val" truthy_builder in
238   let _ = LlvM.build_store truthy_val ret_val_addr truthy_builder in
239   let _ = LlvM.build_br merge_bb truthy_builder in
240
241   (* Falsey basic block: *)
242   let falsey_llvm_var = get_llvm_var false_var falsey_builder in
243   let falsey_val = LlvM.build_call getVal [|falsey_llvm_var; cell_row; cell_col
      |] "falsey_val" falsey_builder in
244   let _ = LlvM.build_store falsey_val ret_val_addr falsey_builder in
245   let _ = LlvM.build_br merge_bb falsey_builder in
246
247   let ret_val = LlvM.build_load ret_val_addr "tern_ret_val" merge_builder in
248   (ret_val, merge_builder)
249 | Selection(expr, sel) ->
250   let (expr_val, expr_builder) = build_expr old_builder expr in
251   let build_rhs_index idx_builder = function
252     Abs(e) ->
253       let (idx_expr_val, next_builder) = build_expr idx_builder e in
254       let rhs_idx_ptr = LlvM.build_alloca base_types.rhs_index_t "idx_ptr"
          next_builder in
255       let _ = (idx_expr_val $> (rhs_idx_ptr, (rhs_index_field_index RhsExprVal))
          ) next_builder in
256       let _ = ((LlvM.const_int base_types.char_t (rhs_index_type_flags_const
          RhsIdxAbs)) $> (rhs_idx_ptr, (rhs_index_field_index RhsIndexType)))
          next_builder in
          (rhs_idx_ptr, next_builder)
257 | Rel(e) ->
258   let (idx_expr_val, next_builder) = build_expr idx_builder e in
259   let rhs_idx_ptr = LlvM.build_alloca base_types.rhs_index_t "idx_ptr"
      next_builder in
260   let _ = (idx_expr_val $> (rhs_idx_ptr, (rhs_index_field_index RhsExprVal))
      ) next_builder in

```

```

262     let _ = ((Llvm.const_int base_types.char_t (rhs_index_type_flags_const
        RhsIdxRel)) $> (rhs_idx_ptr, (rhs_index_field_index RhsIndexType)))
        next_builder in
263     (rhs_idx_ptr, next_builder)
264 | DimensionStart ->
265     let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
        idx_builder in
266     let _ = ((Llvm.const_pointer_null base_types.value_p) $> (rhs_idx_ptr, (
        rhs_index_field_index RhsExprVal))) idx_builder in
267     let _ = ((Llvm.const_int base_types.char_t (rhs_index_type_flags_const
        RhsIdxDimStart)) $> (rhs_idx_ptr, (rhs_index_field_index RhsIndexType))
        ) idx_builder in
268     (rhs_idx_ptr, idx_builder)
269 | DimensionEnd ->
270     let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
        idx_builder in
271     let _ = ((Llvm.const_pointer_null base_types.value_p) $> (rhs_idx_ptr, (
        rhs_index_field_index RhsExprVal))) idx_builder in
272     let _ = ((Llvm.const_int base_types.char_t (rhs_index_type_flags_const
        RhsIdxDimEnd)) $> (rhs_idx_ptr, (rhs_index_field_index RhsIndexType)))
        idx_builder in
273     (rhs_idx_ptr, idx_builder) in
274 let build_rhs_slice slice_builder = function
275     (Some start_idx, Some end_idx) ->
276     let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
        slice_builder in
277     let (start_idx_ptr, next_builder) = build_rhs_index slice_builder
        start_idx in
278     let (end_idx_ptr, last_builder) = build_rhs_index next_builder end_idx in
279     let _ = (start_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
        RhsSliceStartIdx))) last_builder in
280     let _ = (end_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
        RhsSliceEndIdx))) last_builder in
281     (rhs_slice_ptr, last_builder)
282 | (Some single_idx, None) ->
283     let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
        slice_builder in
284     let (single_idx_ptr, last_builder) = build_rhs_index slice_builder
        single_idx in
285     let _ = (single_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
        RhsSliceStartIdx))) last_builder in
286     let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
        rhs_slice_ptr, (rhs_slice_field_index RhsSliceEndIdx))) last_builder in
287     (rhs_slice_ptr, last_builder)
288 | (None, None) ->
289     let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
        slice_builder in
290     let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
        rhs_slice_ptr, (rhs_slice_field_index RhsSliceStartIdx))) slice_builder
        in
291     let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
        rhs_slice_ptr, (rhs_slice_field_index RhsSliceEndIdx))) slice_builder
        in
292     (rhs_slice_ptr, slice_builder)
293 | (None, Some illegal_idx) -> print_endline (string_of_expr exp) ; raise (
    LogicError("This slice should not be grammatically possible")) in

```

```

294 let build_rhs_sel sel_builder = function
295     (Some first_slice, Some second_slice) ->
296     let rhs_selection_ptr = LlvM.build_alloca base_types.rhs_selection_t "
        selection_ptr" sel_builder in
297     let (first_slice_ptr, next_builder) = build_rhs_slice sel_builder
        first_slice in
298     let (second_slice_ptr, last_builder) = build_rhs_slice next_builder
        second_slice in
299     let _ = (first_slice_ptr $> (rhs_selection_ptr, (rhs_selection_field_index
        RhsSelSlice1))) last_builder in
300     let _ = (second_slice_ptr $> (rhs_selection_ptr, (
        rhs_selection_field_index RhsSelSlice2))) last_builder in
301     (rhs_selection_ptr, last_builder)
302 | (Some single_slice, None) ->
303     let rhs_selection_ptr = LlvM.build_alloca base_types.rhs_selection_t "
        selection_ptr" sel_builder in
304     let (single_slice_ptr, last_builder) = build_rhs_slice sel_builder
        single_slice in
305     let _ = (single_slice_ptr $> (rhs_selection_ptr, (
        rhs_selection_field_index RhsSelSlice1))) last_builder in
306     let _ = ((LlvM.const_pointer_null base_types.rhs_slice_p) $> (
        rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice2)))
        last_builder in
307     (rhs_selection_ptr, last_builder)
308 | (None, None) ->
309     let rhs_selection_ptr = LlvM.build_alloca base_types.rhs_selection_t "
        selection_ptr" sel_builder in
310     let _ = ((LlvM.const_pointer_null base_types.rhs_slice_p) $> (
        rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice1)))
        sel_builder in
311     let _ = ((LlvM.const_pointer_null base_types.rhs_slice_p) $> (
        rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice2)))
        sel_builder in
312     (rhs_selection_ptr, sel_builder)
313 | (None, Some illegal_idx) -> print_endline (string_of_expr exp) ; raise (
        LogicError("This selection should not be grammatically possible")) in
314 let (selection_ptr, builder_to_end_all_builders) = build_rhs_sel expr_builder
        sel in
315 (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "
        debug_print_selection") [|selection_ptr|] "" builder_to_end_all_builders in
        *)
316 let ret_val = LlvM.build_call (Hashtbl.find runtime_functions "
        extract_selection") [|expr_val; selection_ptr; cell_row; cell_col|] "
        ret_val" builder_to_end_all_builders in
317 (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print") [|
        ret_val; LlvM.const_pointer_null base_types.char_p|] ""
        builder_to_end_all_builders in *)
318 (ret_val, builder_to_end_all_builders)
319 | Precedence(a,b) -> let (_, new_builder) = build_expr old_builder a in
        build_expr new_builder b
320 | LitString(str) ->
321     let initbod_charptr = LlvM.build_global_stringptr str "initbod_charptr"
        literal_bod in
322     let initbod_val_p = LlvM.build_call (Hashtbl.find runtime_functions "
        new_string") [|initbod_charptr|] "initbod_val_p" literal_bod in
323     let global_val_p_p = LlvM.define_global "global_litstring_p" (LlvM.

```

```

    const_pointer_null base_types.value_p) base_module in
324 let _ = LlvM.build_store initbod_val_p global_val_p_p literal_bod in
325
326 let local_val_p = LlvM.build_load global_val_p_p "local_value_p" old_builder
    in
327 let ret_val = LlvM.build_call (Hashtbl.find runtime_functions "clone_value")
    [|local_val_p|] "ret_val" old_builder in
328 (ret_val, old_builder)
329 | LitRange(rl) ->
330 let num_rows = List.length rl in
331 let num_cols = List.fold_left max 0 (List.map List.length rl) in
332 if num_rows = 1 && num_cols = 1 then build_expr old_builder (List.hd (List.hd
    rl))
333 else
334 let global_val_p_p = LlvM.define_global "global_litrage_p" (LlvM.
    const_pointer_null base_types.value_p) base_module in
335 let initbod_val_p = LlvM.build_malloc base_types.value_t "initbod_val_p"
    literal_bod in
336 let _ = LlvM.build_store initbod_val_p global_val_p_p literal_bod in
337 let _ = (range_type $> (initbod_val_p, (value_field_index Flags)))
    literal_bod in
338 let anonymous_subrange_p = LlvM.build_malloc base_types.subrange_t "
    anonymous_subrange" literal_bod in
339 let _ = (anonymous_subrange_p $> (initbod_val_p, (value_field_index Subrange
    ))) literal_bod in
340
341 let _ = ((LlvM.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
    subrange_field_index BaseOffsetRow))) literal_bod in
342 let _ = ((LlvM.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
    subrange_field_index BaseOffsetCol))) literal_bod in
343 let _ = ((LlvM.const_int base_types.int_t num_rows) $> (anonymous_subrange_p
    , (subrange_field_index SubrangeRows))) literal_bod in
344 let _ = ((LlvM.const_int base_types.int_t num_cols) $> (anonymous_subrange_p
    , (subrange_field_index SubrangeCols))) literal_bod in
345 let anonymous_var_inst_p = LlvM.build_malloc base_types.var_instance_t "
    anonymous_var_inst" literal_bod in
346 let _ = (anonymous_var_inst_p $> (anonymous_subrange_p, (
    subrange_field_index BaseRangePtr))) literal_bod in
347
348 let _ = ((LlvM.const_int base_types.int_t num_rows) $> (anonymous_var_inst_p
    , (var_instance_field_index Rows))) literal_bod in
349 let _ = ((LlvM.const_int base_types.int_t num_cols) $> (anonymous_var_inst_p
    , (var_instance_field_index Cols))) literal_bod in
350 let _ = ((LlvM.const_int base_types.int_t 0) $> (anonymous_var_inst_p, (
    var_instance_field_index NumFormulas))) literal_bod in
351 let _ = ((LlvM.const_pointer_null base_types.resolved_formula_p) $> (
    anonymous_var_inst_p, (var_instance_field_index Formulas))) literal_bod
    in
352 let _ = ((LlvM.const_pointer_null base_types.extend_scope_p) $> (
    anonymous_var_inst_p, (var_instance_field_index Closure))) literal_bod in
353 let vals_array = LlvM.build_array_malloc base_types.value_p (LlvM.const_int
    base_types.int_t (num_rows * num_cols)) "vals_array" literal_bod in
354 let _ = (vals_array $> (anonymous_var_inst_p, (var_instance_field_index
    Values))) literal_bod in
355 let status_array = LlvM.build_array_malloc base_types.char_t (LlvM.const_int
    base_types.int_t (num_rows * num_cols)) "status_array" literal_bod in

```

```

356     let _ = (status_array $> (anonymous_var_inst_p, (var_instance_field_index
357         Status))) literal_bod in
358
359     let get_val_p e = let (vp, _) = build_expr literal_bod e in vp in
360     let val_p_list_list = List.map (fun x -> List.map get_val_p x) rl in
361     let cellnums = zero_until (num_rows * num_cols) in
362     let build_empty x =
363         let emptyval = LlvM.build_malloc base_types.value_t (" " ^ (string_of_int x
364             )) literal_bod in
365         let _ = store_empty emptyval literal_bod in
366         let emptydst = LlvM.build_in_bounds_gep vals_array [|LlvM.const_int
367             base_types.int_t x|] " " literal_bod in
368         let _ = LlvM.build_store emptyval emptydst literal_bod in
369         let statusdst = LlvM.build_in_bounds_gep status_array [|LlvM.const_int
370             base_types.int_t x|] " " literal_bod in
371         let _ = LlvM.build_store (LlvM.const_int base_types.char_t (
372             var_instance_status_flags_index Calculated)) statusdst literal_bod in
373         () in
374     List.iter build_empty cellnums ;
375     let store_val r c realval =
376         let realdst = LlvM.build_in_bounds_gep vals_array [|LlvM.const_int
377             base_types.int_t (r * num_cols + c)|] ("litrangeelemdst" ^ (
378             string_of_int r) ^ "_" ^ (string_of_int c)) literal_bod in
379         let _ = LlvM.build_store realval realdst literal_bod in
380         () in
381     let store_row r cols = List.iteri (fun c v -> store_val r c v) cols in
382     List.iteri store_row val_p_list_list ;
383     (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print") [|
384         initbod_val_p; LlvM.const_pointer_null base_types.char_p|] " " literal_bod
385         in *)
386
387     let local_val_p = LlvM.build_load global_val_p_p "local_value_p" old_builder
388         in
389     (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print") [|
390         local_val_p; LlvM.const_pointer_null base_types.char_p|] " " old_builder
391         in *)
392     let ret_val = LlvM.build_call (Hashtbl.find runtime_functions "clone_value")
393         [|local_val_p|] "ret_val" old_builder in
394     (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print") [|
395         ret_val; LlvM.const_pointer_null base_types.char_p|] " " old_builder in *)
396     (ret_val, old_builder)
397 | Call(fn,exl) -> (*TODO: Call needs to be reviewed. Possibly switch call
398     arguments to value_p*)
399     let build_one_expr (arg_list, intermediate_builder) e =
400         let (arg_val, next_builder) = build_expr intermediate_builder e in
401         (arg_val :: arg_list, next_builder) in
402     let (reversed_arglist, call_builder) = List.fold_left build_one_expr ([],
403         old_builder) exl in
404     let args = Array.of_list (List.rev reversed_arglist) in
405     let result = LlvM.build_call (
406         StringMap.find fn function_llvalues
407     ) args "call_ret_val" call_builder in
408     (result, call_builder)
409 | BinOp(expr1,op,expr2) -> (
410     let (val1, builder1) = build_expr old_builder expr1 in
411     let (val2, int_builder) = build_expr builder1 expr2 in

```

```

396     let bit_shift = (Llvm.const_int base_types.char_t 4) in
397     let expr1_type = (val1 => (value_field_index Flags)) "expr1_type"
        int_builder in
398     let expr2_type = (val2 => (value_field_index Flags)) "expr2_type"
        int_builder in
399     let expr1_type_shifted = Llvm.build_shl expr1_type bit_shift "
        expr_1_type_shifted" int_builder in
400     let combined_type = Llvm.build_add expr1_type_shifted expr2_type "
        combined_type" int_builder in
401     let number_number = Llvm.const_add (Llvm.const_shl number_type bit_shift)
        number_type in
402     let string_string = Llvm.const_add (Llvm.const_shl string_type bit_shift)
        string_type in
403     let empty_empty = Llvm.const_add (Llvm.const_shl empty_type bit_shift)
        empty_type in
404     let range_range = Llvm.const_add (Llvm.const_shl range_type bit_shift)
        range_type in
405     let build_simple_binop oppp int_builder =
406         (let ret_val = Llvm.build_malloc base_types.value_t "binop_minus_ret_val"
            int_builder in
            let _ = Llvm.build_store
                (
                (
                Llvm.const_int
                base_types.char_t
                (value_field_index Empty)
                ) (
                Llvm.build_struct_gep
                ret_val
                (value_field_index Flags)
                ""
                int_builder
                )
                )
            int_builder
            in
            let bailout = (Llvm.append_block context "" form_decl) in
            let bb_bailout = Llvm.builder_at_end context bailout in
            let (numnum_bb, numnum_builder) = make_block "numnum" in
            let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                numnum_builder in
            let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                numnum_builder in
            let numeric_res = oppp numeric_val_1 numeric_val_2 "numeric_res"
                numnum_builder in
            let _ = Llvm.build_store
                numeric_res (
                Llvm.build_struct_gep
                ret_val
                (value_field_index Number)
                ""
                numnum_builder
                )
                numnum_builder in
            let _ = Llvm.build_store
                (
                Llvm.const_int
                base_types.char_t

```

```

440         (value_field_flags_index Number)
441     ) (
442         LlvM.build_struct_gep
443         ret_val
444         (value_field_index Flags)
445         ""
446         numnum_builder
447     )
448     numnum_builder in
449     let _ = LlvM.build_br bailout numnum_builder in
450     let _ = LlvM.build_cond_br (LlvM.build_icmp LlvM.Icmp.Eq combined_type
451         number_number "" int_builder) numnum_bb bailout int_builder in
452     (ret_val, bbailout)
453 )
454 and build_simple_int_binop oppp int_builder =
455     (let ret_val = LlvM.build_malloc base_types.value_t "binop_minus_ret_val"
456         int_builder in
457         let _ = LlvM.build_store
458             (
459                 LlvM.const_int
460                 base_types.char_t
461                 (value_field_flags_index Empty)
462             ) (
463                 LlvM.build_struct_gep
464                 ret_val
465                 (value_field_index Flags)
466                 ""
467                 int_builder
468             )
469         int_builder
470     in
471     let bailout = (LlvM.append_block context "" form_decl) in
472     let bbailout = LlvM.builder_at_end context bailout in
473     let (numnum_bb, numnum_builder) = make_block "numnum" in
474     let roundfl x = LlvM.build_call (Hashtbl.find runtime_functions "lrint
475         ") [|x|] "" numnum_builder in
476     let numeric_val_1 = roundfl ((val1 => (value_field_index Number)) "
477         number_one" numnum_builder) in
478     let numeric_val_2 = roundfl ((val2 => (value_field_index Number)) "
479         number_two" numnum_builder) in
480     let numeric_res = oppp numeric_val_1 numeric_val_2 "numeric_res"
481     numnum_builder in
482     let _ = LlvM.build_store
483         (LlvM.build_sitofp numeric_res base_types.float_t "" numnum_builder
484         )
485         (
486             LlvM.build_struct_gep
487             ret_val
488             (value_field_index Number)
489             ""
490             numnum_builder
491         )
492     numnum_builder in
493     let _ = LlvM.build_store
494         (
495             LlvM.const_int

```

```

489         base_types.char_t
490         (value_field_flags_index Number)
491     ) (
492         Llvm.build_struct_gep
493         ret_val
494         (value_field_index Flags)
495         ""
496         numnum_builder
497     )
498     numnum_builder in
499     let _ = Llvm.build_br bailout numnum_builder in
500     let _ = Llvm.build_cond_br (Llvm.build_icmp Llvm.Icmp.Eq combined_type
501         number_number "" int_builder) numnum_bb bailout int_builder in
502     (ret_val, bbailout)
503 ) in
504 let build_boolean_op numeric_comparator string_comparator int_builder =
505     let ret_val = Llvm.build_malloc base_types.value_t "binop_gt_ret_val"
506         int_builder in
507     let (make_true_bb, make_false_bb, make_empty_bb, merge_builder) =
508         make_truthiness_blocks "binop_eq" ret_val in
509
510     let (numnum_bb, numnum_builder) = make_block "numnum" in
511     let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
512         numnum_builder in
513     let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
514         numnum_builder in
515     let numeric_greater = Llvm.build_fcmp numeric_comparator numeric_val_1
516         numeric_val_2 "numeric_greater" numnum_builder in
517     let _ = Llvm.build_cond_br numeric_greater make_true_bb make_false_bb
518         numnum_builder in
519
520     let (strstr_bb, strstr_builder) = make_block "strstr" in
521     let str_p_1 = (val1 => (value_field_index String)) "string_one"
522         strstr_builder in
523     let str_p_2 = (val2 => (value_field_index String)) "string_two"
524         strstr_builder in
525     let char_p_1 = (str_p_1 => (string_field_index StringCharPtr)) "char_p_one"
526         strstr_builder in
527     let char_p_2 = (str_p_2 => (string_field_index StringCharPtr)) "char_p_two"
528         strstr_builder in
529     let strcmp_result = Llvm.build_call (Hashtbl.find runtime_functions "
530         strcmp") [|char_p_1; char_p_2|] "strcmp_result" strstr_builder in
531     let string_greater = Llvm.build_icmp string_comparator strcmp_result (Llvm
532         .const_null base_types.long_t) "string_greater" strstr_builder in
533     let _ = Llvm.build_cond_br string_greater make_true_bb make_false_bb
534         strstr_builder in
535
536     let switch_inst = Llvm.build_switch combined_type make_empty_bb 2
537         int_builder in (* Incompatible ==> default to empty *)
538     Llvm.add_case switch_inst number_number numnum_bb;
539     Llvm.add_case switch_inst string_string strstr_bb;
540     (ret_val, merge_builder) in
541 match op with
542 | Minus -> build_simple_binop Llvm.build_fsub int_builder
543 | Plus ->
544     let result = Llvm.build_malloc base_types.value_t "" int_builder

```



```

530 and stradd = (Llvm.append_block context "" form_decl)
531 and numadd = (Llvm.append_block context "" form_decl)
532 and bailout = (Llvm.append_block context "" form_decl)
533 and numorstrotorother = (Llvm.append_block context "" form_decl)
534 and strotorother = (Llvm.append_block context "" form_decl)
535 in
536 let bstradd = Llvm.builder_at_end context stradd
537 and bnumadd = Llvm.builder_at_end context numadd
538 and bnumorstrotorother = Llvm.builder_at_end context numorstrotorother
539 and bstrotorother = Llvm.builder_at_end context strotorother
540 and bbailout = Llvm.builder_at_end context bailout
541 and _ = Llvm.build_store (Llvm.const_int base_types.char_t (
    value_field_flags_index Empty)) (Llvm.build_struct_gep result (
    value_field_index Flags) "" int_builder) int_builder
542 in
543 let isnumber = Llvm.build_icmp Llvm.Icmp.Eq (Llvm.build_load (Llvm.
    build_struct_gep val1 (value_field_index Flags) "" bnumorstrotorother)
    "" bnumorstrotorother) (Llvm.const_int base_types.char_t (
    value_field_flags_index Number)) "" bnumorstrotorother
544 and isstring = Llvm.build_icmp Llvm.Icmp.Eq (Llvm.build_load (Llvm.
    build_struct_gep val1 (value_field_index Flags) "" bstrotorother) ""
    bstrotorother) (Llvm.const_int base_types.char_t (
    value_field_flags_index String)) "" bstrotorother
545 and isnumorstring = Llvm.build_icmp Llvm.Icmp.Eq (Llvm.build_load (Llvm.
    build_struct_gep val1 (value_field_index Flags) "" int_builder) ""
    int_builder) (Llvm.build_load (Llvm.build_struct_gep val2 (
    value_field_index Flags) "" int_builder) "" int_builder) ""
    int_builder
546 and _ = Llvm.build_store (Llvm.build_fadd (Llvm.build_load (Llvm.
    build_struct_gep val1 (value_field_index Number) "" bnumadd) ""
    bnumadd) (Llvm.build_load (Llvm.build_struct_gep val2 (
    value_field_index Number) "" bnumadd) "" bnumadd) (Llvm.
    build_struct_gep result (value_field_index Number) "" bnumadd)
    bnumadd
547 and _ = Llvm.build_store (Llvm.const_int base_types.char_t (
    value_field_flags_index Number)) (Llvm.build_struct_gep result (
    value_field_index Flags) "" bnumadd) bnumadd
548 and str1 = Llvm.build_load (Llvm.build_struct_gep val1 (
    value_field_index String) "" bstradd) "" bstradd
549 and str2 = Llvm.build_load (Llvm.build_struct_gep val2 (
    value_field_index String) "" bstradd) "" bstradd
550 and newstr = (Llvm.build_malloc base_types.string_t "" bstradd) in
551 let len1 = Llvm.build_load (Llvm.build_struct_gep str1 (
    string_field_index StringLen) "" bstradd) "" bstradd
552 and len2 = Llvm.build_load (Llvm.build_struct_gep str2 (
    string_field_index StringLen) "" bstradd) "" bstradd
553 and p1 = Llvm.build_load (Llvm.build_struct_gep str1 (string_field_index
    StringCharPtr) "" bstradd) "" bstradd
554 and p2 = Llvm.build_load (Llvm.build_struct_gep str2 (string_field_index
    StringCharPtr) "" bstradd) "" bstradd
555 and dst_char_ptr_ptr = (Llvm.build_struct_gep newstr (string_field_index
    StringCharPtr) "" bstradd)
556 and _ = Llvm.build_store (Llvm.const_int base_types.char_t (
    value_field_flags_index String)) (Llvm.build_struct_gep result (
    value_field_index Flags) "" bstradd) bstradd
557 and _ = Llvm.build_store newstr (Llvm.build_struct_gep result (

```

```

558         value_field_index String) "" bstradd) bstradd in
559     let fullLen = LlvM.build_nsw_add (LlvM.build_nsw_add len1 len2 ""
        bstradd) (LlvM.const_int base_types.long_t 1) "" bstradd
560     and extra_byte2 = (LlvM.build_add len2 (LlvM.const_int base_types.long_t
        1) "" bstradd) in
561     let dst_char = LlvM.build_array_malloc base_types.char_t (LlvM.
        build_trunc fullLen base_types.int_t "" bstradd) "" bstradd in
562     let dst_char2 = LlvM.build_in_bounds_gep dst_char [|len1|] "" bstradd in
563     let _ = LlvM.build_call (Hashtbl.find runtime_functions "llvm.memcpy.
        p0i8.p0i8.i64") [|dst_char; p1; len1; (LlvM.const_int base_types.
        int_t 0); (LlvM.const_int base_types.bool_t 0)|] "" bstradd
564     and _ = LlvM.build_call (Hashtbl.find runtime_functions "llvm.memcpy.
        p0i8.p0i8.i64") [|dst_char2; p2; extra_byte2; (LlvM.const_int
        base_types.int_t 0); (LlvM.const_int base_types.bool_t 0)|] ""
        bstradd
565     and _ = LlvM.build_store dst_char dst_char_ptr_ptr bstradd
566     in
567     let _ = LlvM.build_store (LlvM.build_nsw_add fullLen (LlvM.const_int
        base_types.long_t (-1)) "" bstradd) (LlvM.build_struct_gep newstr (
        string_field_index StringLen) "" bstradd) bstradd
568     in
569     let _ = LlvM.build_cond_br isnumorstring numorstrorother bailout
        int_builder
570     and _ = LlvM.build_cond_br isnumber numadd strorother bnumorstrorother
571     and _ = LlvM.build_cond_br isstring stradd bailout bstrorother
572     and _ = LlvM.build_br bailout bstradd
573     and _ = LlvM.build_br bailout bnumadd
574     in
575     (result, bbailout)
576 | Times -> build_simple_binop LlvM.build_fmulo int_builder
577 | Eq ->
578     (* let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print")
        [|val1; LlvM.build_global_stringptr "Eq operator - value 1" ""
        old_builder|] "" int_builder in
579     let _ = LlvM.build_call (Hashtbl.find runtime_functions "debug_print") [|
        val2; LlvM.build_global_stringptr "Eq operator - value 2" ""
        old_builder|] "" int_builder in *)
580     let ret_val = LlvM.build_malloc base_types.value_t "binop_eq_ret_val"
        int_builder in
581     let (make_true_bb, make_false_bb, _, merge_builder) =
        make_truthiness_blocks "binop_eq" ret_val in
582     let (numnum_bb, numnum_builder) = make_block "numnum" in
583     let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
        numnum_builder in
584     let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
        numnum_builder in
585     let numeric_equality = LlvM.build_fcmp LlvM.Fcmp.Oeq numeric_val_1
        numeric_val_2 "numeric_equality" numnum_builder in
586     let _ = LlvM.build_cond_br numeric_equality make_true_bb make_false_bb
        numnum_builder in
587     let (strsr_bb, strsr_builder) = make_block "strsr" in
588     let str_p_1 = (val1 => (value_field_index String)) "string_one"
        strsr_builder in
589     let str_p_2 = (val2 => (value_field_index String)) "string_two"

```

```

        strstr_builder in
591   let char_p_1 = (str_p_1 => (string_field_index StringCharPtr)) "char_p_one
        " strstr_builder in
592   let char_p_2 = (str_p_2 => (string_field_index StringCharPtr)) "char_p_two
        " strstr_builder in
593   let strcmp_result = LlvM.build_call (Hashtbl.find runtime_functions "
        strcmp") [|char_p_1; char_p_2|] "strcmp_result" strstr_builder in
594   let string_equality = LlvM.build_icmp LlvM.Icmp.Eq strcmp_result (LlvM.
        const_null base_types.long_t) "string_equality" strstr_builder in
595   let _ = LlvM.build_cond_br string_equality make_true_bb make_false_bb
        strstr_builder in
596
597   let (rngrng_bb, rngrng_builder) = make_block "rngrng" in
598   (* TODO: Make this case work *)
599   let eqt = LlvM.build_is_not_null (LlvM.build_call (Hashtbl.find
        runtime_functions "rg_eq") [|val1; val2|] "" rngrng_builder) ""
        rngrng_builder in
600   let _ = LlvM.build_cond_br eqt make_true_bb make_false_bb rngrng_builder
        in
601
602   let switch_inst = LlvM.build_switch combined_type make_false_bb 4
        int_builder in (* Incompatible ==> default to false *)
603   LlvM.add_case switch_inst number_number numnum_bb;
604   LlvM.add_case switch_inst string_string strstr_bb;
605   LlvM.add_case switch_inst range_range rngrng_bb;
606   LlvM.add_case switch_inst empty_empty make_true_bb; (* Nothing to check in
        this case, just return true *)
607   (ret_val, merge_builder)
608 | Gt -> build_boolean_op LlvM.Fcmp.Ogt LlvM.Icmp.Sgt int_builder
609 | GtEq -> build_boolean_op LlvM.Fcmp.Oge LlvM.Icmp.Sge int_builder
610 | Lt -> build_boolean_op LlvM.Fcmp.Olt LlvM.Icmp.Slt int_builder
611 | LtEq -> build_boolean_op LlvM.Fcmp.Ole LlvM.Icmp.Sle int_builder
612 | LogAnd | LogOr -> raise (TransformedAway("&& and || should have been
        transformed into a short-circuit ternary expression! Error in the
        following expression:\n" ^ string_of_expr exp))
613 | Divide-> build_simple_binop LlvM.build_fdiv int_builder
614 | Mod-> build_simple_binop LlvM.build_frem int_builder
615 | Pow-> (
616   let powcall numeric_val_1 numeric_val_2 valname b =
617     LlvM.build_call (Hashtbl.find runtime_functions "pow") [|numeric_val_1;
        numeric_val_2|] "" b in
618   build_simple_binop powcall int_builder)
619 | LShift-> build_simple_int_binop LlvM.build_shl int_builder
620 | RShift-> build_simple_int_binop LlvM.build_lshr int_builder
621 | BitOr-> build_simple_int_binop LlvM.build_or int_builder
622 | BitAnd-> build_simple_int_binop LlvM.build_and int_builder
623 | BitXor-> build_simple_int_binop LlvM.build_xor int_builder
624 )
625 | UnOp(SizeOf,expr) ->
626   let ret_val = LlvM.build_malloc base_types.value_t "unop_size_ret_val"
        old_builder in
627
628   (* TODO: We actually have to keep track of these anonymous objects somewhere
        so we can free them *)
629   let _ = (range_type $> (ret_val, (value_field_index Flags))) old_builder in
630   let anonymous_subrange_p = LlvM.build_malloc base_types.subrange_t "

```

```

        anonymous_subrange" old_builder in
631 let _ = (anonymous_subrange_p $> (ret_val, (value_field_index Subrange)))
        old_builder in
632
633 let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
        subrange_field_index BaseOffsetRow))) old_builder in
634 let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
        subrange_field_index BaseOffsetCol))) old_builder in
635 let _ = ((Llvm.const_int base_types.int_t 1) $> (anonymous_subrange_p, (
        subrange_field_index SubrangeRows))) old_builder in
636 let _ = ((Llvm.const_int base_types.int_t 2) $> (anonymous_subrange_p, (
        subrange_field_index SubrangeCols))) old_builder in
637 let anonymous_var_inst_p = Llvm.build_malloc base_types.var_instance_t "
        anonymous_var_inst" old_builder in
638 let _ = (anonymous_var_inst_p $> (anonymous_subrange_p, (subrange_field_index
        BaseRangePtr))) old_builder in
639
640 let _ = ((Llvm.const_int base_types.int_t 1) $> (anonymous_var_inst_p, (
        var_instance_field_index Rows))) old_builder in
641 let _ = ((Llvm.const_int base_types.int_t 2) $> (anonymous_var_inst_p, (
        var_instance_field_index Cols))) old_builder in
642 let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_var_inst_p, (
        var_instance_field_index NumFormulas))) old_builder in
643 let _ = ((Llvm.const_pointer_null base_types.resolved_formula_p) $> (
        anonymous_var_inst_p, (var_instance_field_index Formulas))) old_builder in
644 let _ = ((Llvm.const_pointer_null base_types.extend_scope_p) $> (
        anonymous_var_inst_p, (var_instance_field_index Closure))) old_builder in
645 let num_rows_val = Llvm.build_malloc base_types.value_t "num_rows_val"
        old_builder in
646 let num_cols_val = Llvm.build_malloc base_types.value_t "num_cols_val"
        old_builder in
647 let vals_array = Llvm.build_array_malloc base_types.value_p (Llvm.const_int
        base_types.int_t 2) "vals_array" old_builder in
648 let _ = (vals_array $> (anonymous_var_inst_p, (var_instance_field_index Values
        ))) old_builder in
649 let _ = Llvm.build_store num_rows_val (Llvm.build_in_bounds_gep vals_array [|
        Llvm.const_int base_types.int_t 0|] "" old_builder) old_builder in
650 let _ = Llvm.build_store num_cols_val (Llvm.build_in_bounds_gep vals_array [|
        Llvm.const_int base_types.int_t 1|] "" old_builder) old_builder in
651 let status_array = Llvm.build_array_malloc base_types.char_t (Llvm.const_int
        base_types.int_t 2) "status_array" old_builder in
652 let _ = (status_array $> (anonymous_var_inst_p, (var_instance_field_index
        Status))) old_builder in
653 let _ = Llvm.build_store (Llvm.const_int base_types.char_t (
        var_instance_status_flags_index Calculated)) (Llvm.build_in_bounds_gep
        status_array [|Llvm.const_int base_types.int_t 0|] "" old_builder)
        old_builder in
654 let _ = Llvm.build_store (Llvm.const_int base_types.char_t (
        var_instance_status_flags_index Calculated)) (Llvm.build_in_bounds_gep
        status_array [|Llvm.const_int base_types.int_t 1|] "" old_builder)
        old_builder in
655
656 let (expr_val, expr_builder) = build_expr old_builder expr in
657 let val_flags = (expr_val => (value_field_index Flags)) "val_flags"
        expr_builder in
658 let is_subrange = Llvm.build_icmp Llvm.Icmp.Eq val_flags range_type "

```

```

        is_subrange" expr_builder in
659
660     let (merge_bb, merge_builder) = make_block "merge" in
661
662     let (primitive_bb, primitive_builder) = make_block "primitive" in
663     let _ = store_number num_rows_val primitive_builder (Llvm.const_float
        base_types.float_t 1.0) in
664     let _ = store_number num_cols_val primitive_builder (Llvm.const_float
        base_types.float_t 1.0) in
665     let _ = Llvm.build_br merge_bb primitive_builder in
666
667     let (subrange_bb, subrange_builder) = make_block "subrange" in
668     let subrange_ptr = (expr_val => (value_field_index Subrange)) "subrange_ptr"
        subrange_builder in
669     let rows_as_int = (subrange_ptr => (subrange_field_index SubrangeRows)) "
        rows_as_int" subrange_builder in
670     let cols_as_int = (subrange_ptr => (subrange_field_index SubrangeCols)) "
        cols_as_int" subrange_builder in
671     let rows_as_float = Llvm.build_sitofp rows_as_int base_types.float_t "
        rows_as_float" subrange_builder in
672     let cols_as_float = Llvm.build_sitofp cols_as_int base_types.float_t "
        cols_as_float" subrange_builder in
673     let _ = store_number num_rows_val subrange_builder rows_as_float in
674     let _ = store_number num_cols_val subrange_builder cols_as_float in
675     let _ = Llvm.build_br merge_bb subrange_builder in
676
677     let _ = Llvm.build_cond_br is_subrange subrange_bb primitive_bb expr_builder
        in
678     (ret_val, merge_builder)
679 | UnOp(Truthy, expr) ->
680     let ret_val = Llvm.build_malloc base_types.value_t "unop_truthy_ret_val"
        old_builder in
681     let (expr_val, expr_builder) = build_expr old_builder expr in
682
683     let (truthy_bb, falsey_bb, empty_bb, merge_builder) = make_truthiness_blocks "
        unop_truthy" ret_val in
684
685     let expr_flags = (expr_val => (value_field_index Flags)) "expr_flags"
        expr_builder in
686     let is_empty_bool = (Llvm.build_icmp Llvm.Icmp.Eq expr_flags (Llvm.const_int
        base_types.flags_t (value_field_flags_index Empty)) "is_empty_bool"
        expr_builder) in
687     let is_empty = Llvm.build_zext is_empty_bool base_types.char_t "is_empty"
        expr_builder in
688     let is_empty_two = Llvm.build_shl is_empty (Llvm.const_int base_types.char_t
        1) "is_empty_two" expr_builder in
689     let is_number = Llvm.build_icmp Llvm.Icmp.Eq expr_flags (Llvm.const_int
        base_types.flags_t (value_field_flags_index Number)) "is_number"
        expr_builder in
690     let the_number = (expr_val => (value_field_index Number)) "the_number"
        expr_builder in
691     let is_zero = Llvm.build_fcmp Llvm.Fcmp.Oeq the_number (Llvm.const_float
        base_types.number_t 0.0) "is_zero" expr_builder in
692     let is_numeric_zero_bool = Llvm.build_and is_zero is_number "
        is_numeric_zero_bool" expr_builder in
693     let is_numeric_zero = Llvm.build_zext is_numeric_zero_bool base_types.char_t "

```

```

        is_numeric_zero" expr_builder in
694   let switch_num = LlvM.build_add is_empty_two is_numeric_zero "switch_num"
        expr_builder in
695   let switch_inst = LlvM.build_switch switch_num empty_bb 2 expr_builder in
696   LlvM.add_case switch_inst (LlvM.const_int base_types.char_t 0) truthy_bb; (*
        empty << 1 + is_zero == 0 ==> truthy *)
697   LlvM.add_case switch_inst (LlvM.const_int base_types.char_t 1) falsey_bb; (*
        empty << 1 + is_zero == 1 ==> falsey *)
698   (ret_val, merge_builder)
699 | UnOp(LogNot, expr) ->
700   let (truth_val, truth_builder) = build_expr old_builder (UnOp(Truthy, expr))
        in
701   let the_number = (truth_val => (value_field_index Number)) "the_number"
        truth_builder in
702   let not_the_number = LlvM.build_fsub (LlvM.const_float base_types.float_t 1.0)
        the_number "not_the_number" truth_builder in
703   let sp = LlvM.build_struct_gep truth_val (value_field_index Number) "
        num_pointer" truth_builder in
704   let _ = LlvM.build_store not_the_number sp truth_builder in
705   (truth_val, truth_builder)
706 | UnOp(Neg, expr) ->
707   let ret_val = LlvM.build_malloc base_types.value_t "unop_truthy_ret_val"
        old_builder in
708   let _ = store_empty ret_val old_builder in
709   let (expr_val, expr_builder) = build_expr old_builder expr in
710   let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
        expr_builder in
711   let is_number = LlvM.build_icmp LlvM.Icmp.Eq expr_type number_type "is_number"
        expr_builder in
712   let (finish_bb, finish_builder) = make_block "finish" in
713
714   let (number_bb, number_builder) = make_block "number" in
715   let the_number = (expr_val => (value_field_index Number)) "the_number"
        number_builder in
716   let minus_the_number = LlvM.build_fneg the_number "minus_the_number"
        number_builder in
717   let _ = store_number ret_val number_builder minus_the_number in
718   let _ = LlvM.build_br finish_bb number_builder in
719
720   let _ = LlvM.build_cond_br is_number number_bb finish_bb expr_builder in
721   (ret_val, finish_builder)
722 | UnOp(BitNot, expr) ->
723   let ret_val = LlvM.build_malloc base_types.value_t "unop_truthy_ret_val"
        old_builder in
724   let (expr_val, expr_builder) = build_expr old_builder expr in
725
726   let (numnum_bb, numnum_builder) = make_block "numnum" in
727   let (make_empty_bb, make_empty_builder) = make_block (" " ^ "_empty") in
728   let (finish_bb, finish_builder) = make_block "finish" in
729
730   let _ = store_empty ret_val make_empty_builder in
731   let _ = LlvM.build_br finish_bb make_empty_builder in
732
733   let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
        expr_builder in
734   let is_number = LlvM.build_icmp LlvM.Icmp.Eq expr_type number_type "is_number"

```

```

    expr_builder in
735     let _ = LlvM.build_cond_br is_number numnum_bb make_empty_bb expr_builder in
736
737     let expr_num = LlvM.build_call (Hashtbl.find runtime_functions "lrint") [|((
        expr_val => (value_field_index Number)) "expr_type" numnum_builder)|] " "
        numnum_builder in
738     let _ = store_number ret_val numnum_builder (LlvM.build_sitofp (LlvM.build_not
        expr_num " " numnum_builder) base_types.float_t " " numnum_builder) in
739     let _ = LlvM.build_br finish_bb numnum_builder in
740
741     (ret_val, finish_builder)
742 | UnOp(TypeOf, expr) ->
743     let (expr_val, expr_builder) = build_expr old_builder expr in
744     let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
        expr_builder in
745     let vp_to_clone_loc = LlvM.build_in_bounds_gep array_of_typeof_val_ptrs [|LlvM
        .const_int base_types.int_t 0; expr_type|] ("vp_to_clone_log") expr_builder
        in
746     let vp_to_clone = LlvM.build_load vp_to_clone_loc "vp_to_clone" expr_builder
        in
747     let ret_val = LlvM.build_call (Hashtbl.find runtime_functions "clone_value")
        [|vp_to_clone|] "typeof_ret_val" expr_builder in
748     (ret_val, expr_builder)
749 | UnOp(Row, _) ->
750     let row_as_int = cell_row in
751     let row_as_float = LlvM.build_sitofp row_as_int base_types.float_t "
        row_as_float" old_builder in
752     let ret_val = LlvM.build_malloc base_types.value_t "ret_val" old_builder in
753     let _ = store_number ret_val old_builder row_as_float in
754     (ret_val, old_builder)
755 | UnOp(Column, _) ->
756     let col_as_int = cell_col in
757     let col_as_float = LlvM.build_sitofp col_as_int base_types.float_t "
        col_as_float" old_builder in
758     let ret_val = LlvM.build_malloc base_types.value_t "ret_val" old_builder in
759     let _ = store_number ret_val old_builder col_as_float in
760     (ret_val, old_builder)
761 | Switch(_,_,_) | Ternary(_,_,_) -> raise(TransformedAway("These expressions
        should have been transformed away")) in
762     (* | unknown_expr -> print_endline (string_of_expr unknown_expr);raise
        NotImplemented in *)
763     let (ret_value_p, final_builder) = build_expr builder_at_top formula_expr in
764     let _ = LlvM.build_ret ret_value_p final_builder in
765     form_decl in
766
767     (*build formula creates a formula declaration in a separate method from the function
        it belongs to*)
768 let build_formula (varname, idx) formula_array element symbols =
769     let storage_addr = LlvM.build_in_bounds_gep formula_array [|LlvM.const_int
        base_types.int_t idx|] " " init_bod in
770     let getStarts = function (* Not really just for starts *)
771         Abs(LitInt(1)) | Abs(LitInt(0)) | DimensionStart | DimensionEnd -> (1, -1)
772     | Abs(Id(s)) ->
773         (match StringMap.find s symbols with
774             LocalVariable(i) | GlobalVariable(i) -> (0, i)
775             | _ -> raise(TransformedAway("Error in " ^ varname ^ ": The LHS expresssions

```

```

        should always either have dimension length 1 or be the name of a variable
        in their own scope.)))
776 | _ -> print_endline ("Error in " ^ varname ^ " formula number " ^ string_of_int
        idx); raise (LogicError ("Something wrong with the index of formula: " ^
        string_of_formula element)) in
777 let getEnds = function
778   Some x -> let (b, c) = getStarts x in (b, c, 0)
779 | None -> (0, -1, 1) in
780 let (fromStartRow, rowStartVarnum) = getStarts element.formula_row_start in
781 let (fromStartCol, colStartVarnum) = getStarts element.formula_col_start in
782 let (toEndRow, rowEndVarnum, isSingleRow) = getEnds element.formula_row_end in
783 let (toEndCol, colEndVarnum, isSingleCol) = getEnds element.formula_col_end in
784
785 let _ = LlvM.build_store (LlvM.const_int base_types.char_t fromStartRow) (LlvM.
    build_struct_gep storage_addr (formula_field_index FromFirstRow) "" init_bod)
    init_bod in
786 let _ = LlvM.build_store (LlvM.const_int base_types.int_t rowStartVarnum) (LlvM.
    build_struct_gep storage_addr (formula_field_index RowStartNum) "" init_bod)
    init_bod in
787 let _ = LlvM.build_store (LlvM.const_int base_types.char_t toEndRow) (LlvM.
    build_struct_gep storage_addr (formula_field_index ToLastRow) "" init_bod)
    init_bod in
788 let _ = LlvM.build_store (LlvM.const_int base_types.int_t rowEndVarnum) (LlvM.
    build_struct_gep storage_addr (formula_field_index RowEndNum) "" init_bod)
    init_bod in
789 let _ = LlvM.build_store (LlvM.const_int base_types.char_t isSingleRow) (LlvM.
    build_struct_gep storage_addr (formula_field_index IsSingleRow) "" init_bod)
    init_bod in
790
791 let _ = LlvM.build_store (LlvM.const_int base_types.char_t fromStartCol) (LlvM.
    build_struct_gep storage_addr (formula_field_index FromFirstCols) "" init_bod)
    init_bod in
792 let _ = LlvM.build_store (LlvM.const_int base_types.int_t colStartVarnum) (LlvM.
    build_struct_gep storage_addr (formula_field_index ColStartNum) "" init_bod)
    init_bod in
793 let _ = LlvM.build_store (LlvM.const_int base_types.char_t toEndCol) (LlvM.
    build_struct_gep storage_addr (formula_field_index ToLastCol) "" init_bod)
    init_bod in
794 let _ = LlvM.build_store (LlvM.const_int base_types.int_t colEndVarnum) (LlvM.
    build_struct_gep storage_addr (formula_field_index ColEndNum) "" init_bod)
    init_bod in
795 let _ = LlvM.build_store (LlvM.const_int base_types.char_t isSingleCol) (LlvM.
    build_struct_gep storage_addr (formula_field_index IsSingleCol) "" init_bod)
    init_bod in
796
797 let form_decl = build_formula_function (varname, idx) symbols element.formula_expr
    in
798 let _ = LlvM.build_store form_decl (LlvM.build_struct_gep storage_addr (
    formula_field_index FormulaCall) "" init_bod) init_bod in
799 () in
800
801 (* Builds a var_defn struct for each variable *)
802 let build_var_defn defn varname va symbols =
803   let numForm = List.length va.var_formulas in
804   let formulas = LlvM.build_array_malloc base_types.formula_t (LlvM.const_int
    base_types.int_t numForm) "" init_bod in

```



```

805 (*getDefn simply looks up the correct definition for a dimension declaration of a
      variable. Note that currently it is ambiguous whether it is a variable or a
      literal. TODO: consider negative numbers*)
806 let getDefn = function
807   DimId(a) -> (match StringMap.find a symbols with LocalVariable(i) -> i |
      GlobalVariable(i) -> i | _ -> raise(TransformedAway("Error in " ^ varname ^
      ": The LHS expressions should always either have dimension length 1 or be
      the name of a variable in their own scope.")))
808   | DimOneByOne -> 1 in
809 let _ = (match va.var_rows with
810   DimOneByOne -> LlvM.build_store (LlvM.const_int base_types.char_t 1) (LlvM.
      build_struct_gep defn (var_defn_field_index OneByOne) "" init_bod)
      init_bod
811   | DimId(a) -> (
812     let _ = LlvM.build_store (LlvM.const_int base_types.char_t 0) (LlvM.
      build_struct_gep defn (var_defn_field_index OneByOne) "" init_bod)
      init_bod in ();
813     let _ = LlvM.build_store (LlvM.const_int base_types.int_t (getDefn va.
      var_rows)) (LlvM.build_struct_gep defn (var_defn_field_index Rows) ""
      init_bod) init_bod in ();
814     LlvM.build_store (LlvM.const_int base_types.int_t (getDefn va.var_cols)) (
      LlvM.build_struct_gep defn (var_defn_field_index Cols) "" init_bod)
      init_bod
815   )
816   ) in
817 let _ = LlvM.build_store (LlvM.const_int base_types.int_t numForm) (LlvM.
      build_struct_gep defn (var_defn_field_index NumFormulas) "" init_bod) init_bod
818 and _ = LlvM.build_store formulas (LlvM.build_struct_gep defn (
      var_defn_field_index Formulas) "" init_bod) init_bod
819 and _ = LlvM.build_store (LlvM.build_global_stringptr varname "" init_bod) (LlvM.
      build_struct_gep defn (var_defn_field_index VarName) "" init_bod) init_bod in
820 List.iteri (fun idx elem -> build_formula (varname, idx) formulas elem symbols) va
      .var_formulas in
821
822 (* Creates a scope object and inserts the necessary instructions into main to
      populate the var_defns, and
823 * into the function specified by builder to populate the scope object. *)
824 let build_scope_obj
825   fname (* The function name, or "globals" *)
826   symbols (* The symbols to use when creating the functions *)
827   vars (* The variables to build definitions and formula-functions for *)
828   static_location_ptr (* The copy of the global pointer used in main *)
829   var_defns_loc (* The copy of the global pointer used in the local function *)
830   num_params (* How many parameters the function takes *)
831   builder (* The LLVM builder for the local function *)
832   =
833   let cardinal = LlvM.const_int base_types.int_t (StringMap.cardinal vars) in
834   let build_var_defns =
835     let static_var_defns = LlvM.build_array_malloc base_types.var_defn_t cardinal (
      fname ^ "_static_var_defns") init_bod in
836     let _ = LlvM.build_store static_var_defns static_location_ptr init_bod in
837     let add_variable varname va (sm, count) =
838       let fullname = fname ^ "_" ^ varname in
839       let defn = (LlvM.build_in_bounds_gep static_var_defns [|LlvM.const_int
      base_types.int_t count|] (fullname ^ "_defn") init_bod) in
840       let _ = build_var_defn defn fullname va symbols in

```

```

841     (StringMap.add varname count sm, count + 1) in
842     ignore (StringMap.fold add_variable vars (StringMap.empty, 0)) in
843
844     let var_defns = LlvM.build_load var_defns_loc (fname ^ "_global_defn_ptr_loc")
      builder in
845     let var_insts = LlvM.build_array_malloc base_types.var_instance_p cardinal "
      var_insts" builder in
846     let scope_obj = LlvM.build_malloc base_types.extend_scope_t "scope_obj" builder in
847
848     (*Store variable definition and instance*)
849     let _ = LlvM.build_store var_defns (LlvM.build_struct_gep scope_obj (
      scope_field_type_index VarDefn) "" builder) builder in
850     let _ = LlvM.build_store var_insts (LlvM.build_struct_gep scope_obj (
      scope_field_type_index VarInst) "" builder) builder in
851     let _ = LlvM.build_store cardinal (LlvM.build_struct_gep scope_obj (
      scope_field_type_index VarNum) "" builder) builder in
852     let _ = LlvM.build_store (LlvM.const_int base_types.int_t 0) (LlvM.
      build_struct_gep scope_obj (scope_field_type_index ScopeRefCount) "" builder)
      builder in
853     let paramarray = if num_params > 0 then LlvM.build_array_malloc base_types.value_p
      (LlvM.const_int base_types.int_t num_params) "paramarray" builder else LlvM.
      const_pointer_null (LlvM.pointer_type base_types.value_p) in
854     let _ = LlvM.build_store paramarray (LlvM.build_struct_gep scope_obj (
      scope_field_type_index FunctionParams) "" builder) builder in
855     let copy_fn_arg i =
856       let param_addr = LlvM.build_in_bounds_gep paramarray [|LlvM.const_int base_types
        .int_t i|] (fname ^ "_param_" ^ string_of_int i ^ "_loc") builder in
857       ignore (LlvM.build_store (LlvM.param (StringMap.find fname function_llvalues) i)
        param_addr builder) in
858     List.iter copy_fn_arg (zero_until num_params);
859     let _ = LlvM.build_call (Hashtbl.find runtime_functions "null_init") [|scope_obj|]
      "" builder in
860     build_var_defns ; scope_obj in
861 (* End of build_scope_obj *)
862
863 let build_function fname (fn_def, fn_llvalue) =
864   (* Build the symbol table for this function *)
865   let symbols = create_symbol_table global_symbols fn_def in
866   let fn_idx = match StringMap.find fname extend_fn_numbers with ExtendFunction(i)
     -> i | _ -> raise (LogicError(fname ^ " not in function table")) in
867   let builder = LlvM.builder_at_end context (LlvM.entry_block fn_llvalue) in
868   let static_location_ptr = LlvM.build_in_bounds_gep array_of_vardefn_ptrs [|LlvM.
     const_int base_types.int_t 0; LlvM.const_int base_types.int_t fn_idx|] (fname ^
     "_global_defn_ptr") init_bod in
869   let var_defns_loc = LlvM.build_in_bounds_gep array_of_vardefn_ptrs [|LlvM.
     const_int base_types.int_t 0; LlvM.const_int base_types.int_t fn_idx|] (fname ^
     "_local_defn_ptr") builder in
870   let scope_obj = build_scope_obj fname symbols fn_def.func_body static_location_ptr
     var_defns_loc (List.length fn_def.func_params) builder in
871   let get_special_val special_name = function
872     Id(s) -> (match (try StringMap.find s symbols with Not_found -> raise(
      LogicError("Something went wrong with your semantic analysis - " ^ s ^ "
      not found")))) with
      LocalVariable(i) ->
873       let llvm_var = LlvM.build_call getVar [|scope_obj; LlvM.const_int
        base_types.int_t i|] (special_name ^ "_var") builder in

```

```

875         Llvml.build_call getVal [|llvm_var; Llvml.const_int base_types.int_t 0; Llvml
            .const_int base_types.int_t 0|] (special_name ^ "_val") builder
876         | _ -> raise(TransformedAway("Error in " ^ fname ^ ": The " ^ special_name ^
            " value should always have been transformed into a local variable")))
877         | _ -> raise(TransformedAway("Error in " ^ fname ^ ": The " ^ special_name ^ "
            value should always have been transformed into a local variable")) in
878     let assert_val = get_special_val "assert" (List.hd fn_def.func_asserts) in
879     let _ = Llvml.build_call (Hashtbl.find runtime_functions "verify_assert") [|
        assert_val; Llvml.build_global_stringptr fname "" builder|] "" builder in
880     let ret_val = get_special_val "return" (snd fn_def.func_ret_val) in
881     let _ = Llvml.build_ret ret_val builder in () in
882 (* End of build_function *)
883
884 (* Build the global scope object *)
885 let vardefn_p_p = Llvml.build_alloca base_types.var_defn_p "v_p_p" init_bod in
886 let global_scope_obj = build_scope_obj "globals" global_symbols globals vardefn_p_p
    vardefn_p_p 0 init_bod in
887 let _ = Llvml.build_call (Hashtbl.find runtime_functions "incStack") [|]| "" init_bod
    in
888 let _ = Llvml.build_store global_scope_obj global_scope_loc init_bod in
889
890 (*iterates over function definitions*)
891 StringMap.iter build_function extend_functions ;
892
893 (* Define the LLVM entry point for the program *)
894 let extend_entry_point = StringMap.find "main" function_llvalues in
895 let _ = Llvml.build_ret_void init_bod in
896 let _ = Llvml.build_ret_void literal_bod in
897 let _ = Llvml.build_call init_def [|]| "" main_bod in
898 let _ = Llvml.build_call literal_def [|]| "" main_bod in
899 let cmd_line_args = Llvml.build_call (Hashtbl.find runtime_functions "
    box_command_line_args") [|Llvml.param main_def 0; Llvml.param main_def 1|] ""
    cmd_line_args" main_bod in
900 let _ = Llvml.build_call extend_entry_point [|cmd_line_args|] "" main_bod in
901 let _ = Llvml.build_ret (Llvml.const_int base_types.int_t 0) main_bod in
902
903 base_module
904
905 let build_this ast_mapped =
906     let modu = (translate ast_mapped) in
907     let _ = Llvml_analysis.assert_valid_module modu in
908     modu

```

## 9.8 linker.ml

```

1  (* ns3158 *)
2  module StringSet = Set.Make(String)
3  let link xtndOut ast compiler outputFile =
4      let tmpFilenameLL = Filename.temp_file "" ".ll"
5      and tmpFilenameC = Filename.temp_file "" ".o"
6      and getExterns (_,_,extern) =
7          StringSet.elements
8              (Ast.StringMap.fold
9                  (fun key value store -> StringSet.add value.Ast.extern_fn_libname store)
10                     extern

```

```

11     StringSet.empty) in
12   let tmpChan = open_out tmpFilenameLL in
13   output_string tmpChan xtndOut; close_out tmpChan;
14   let call1 = (String.concat " " ("llc-3.8" :: "-filetype=obj" :: tmpFilenameLL :: "-o"
    " :: tmpFilenameC :: []))
15   and call2 = (String.concat " " (compiler :: "-o" :: outputFile :: tmpFilenameC :: (
    getExterns ast))) ^ "-lm" in
16   let resc1 = Sys.command call1 in
17   if resc1 == 0 then (
18     Sys.remove tmpFilenameLL;
19     let resc2 = Sys.command call2 in
20     Sys.remove tmpFilenameC;
21     if resc2 == 0 then () else raise Not_found
22   )
23   else (Sys.remove tmpFilenameC; raise Not_found)

```

## 9.9 main.ml

```

1  (* jss2272 *)
2
3  open Ast;;
4
5  let print_ast = ref false
6  let compile_ast = ref false
7  let link = ref false
8  let output = ref "./out"
9  let compiler = ref "gcc"
10 let working_dir = ref "."
11
12 let the_ast = ref (StringMap.empty, StringMap.empty, StringMap.empty)
13 let just_one_please = ref false
14
15 let speclist = [
16     ("-p", Arg.Set print_ast, "Print the AST");
17     ("-c", Arg.Set compile_ast, "Compile the program");
18     ("-l", Arg.Set link, "Link the program");
19     ("-cc", Arg.Set_string compiler, "Compiler to use");
20     ("-o", Arg.Set_string output, "Location to output to");
21     ("-w", Arg.Set_string working_dir, "Working directory");
22 ]
23
24 let usage_message = "Welcome to Extend!\n\nUsage: extend <options> <source-file>\n\
    nOptions are:"
25
26 let parse_ast filename =
27   if !just_one_please
28   then print_endline "Any files after the first one are ignored."
29   else just_one_please := true ; the_ast := (Transform.create_ast filename);;
30
31 Arg.parse speclist parse_ast usage_message;
32 Sys.chdir !working_dir;
33 if not !just_one_please then Arg.usage speclist usage_message else ();
34 if !print_ast then print_endline (string_of_program !the_ast) else ();
35 if !compile_ast then
36   let compiled = (Llvm.string_of_llmodule (Codegen.translate !the_ast))

```

```

37     in
38     if not (!link) then print_endline compiled
39     else Linker.link compiled !the_ast !compiler !output
40 else ();

```

## 9.10 lib.c

```

1  /* jss2272 ns3158 isk2108 */
2
3  #include<stdio.h>
4  #include<stdlib.h>
5  #include<math.h>
6  #include<string.h>
7  #include<stdbool.h>
8  #include "../lib/gdchart0.94b/gdc.h"
9  #include "../lib/gdchart0.94b/gdchart.h"
10 /* #include <sys/time.h> */
11 #include <time.h>
12 #include "runtime.h"
13
14 /* Value type */
15 #define FLAG_EMPTY 0
16 #define FLAG_NUMBER 1
17 #define FLAG_STRING 2
18 #define FLAG_SUBRANGE 3
19
20 /* Status flag */
21 #define CALCULATED 2
22 #define IN_PROGRESS 4
23
24 #define MAX_FILES 255
25 FILE *open_files[1 + MAX_FILES] = {NULL};
26 int open_num_files = 0;
27
28 #define EXPOSE_MATH_FUNC(name) value_p extend_##name(value_p a){if(!assertSingleNumber
    (a)) return new_val();double val = name(a->numericVal);return new_number(val);}
29 EXPOSE_MATH_FUNC(sin)
30 EXPOSE_MATH_FUNC(cos)
31 EXPOSE_MATH_FUNC(tan)
32 EXPOSE_MATH_FUNC(acos)
33 EXPOSE_MATH_FUNC(asin)
34 EXPOSE_MATH_FUNC(atan)
35 EXPOSE_MATH_FUNC(sinh)
36 EXPOSE_MATH_FUNC(cosh)
37 EXPOSE_MATH_FUNC(tanh)
38 EXPOSE_MATH_FUNC(exp)
39 EXPOSE_MATH_FUNC(log)
40 EXPOSE_MATH_FUNC(log10)
41 EXPOSE_MATH_FUNC(sqrt)
42 EXPOSE_MATH_FUNC(ceil)
43 EXPOSE_MATH_FUNC(fabs)
44 EXPOSE_MATH_FUNC(floor)
45
46 value_p extend_round(value_p num, value_p number_of_digits) {
47     if (!assertSingleNumber(num) || !assertSingleNumber(number_of_digits)) return

```

```

        new_val();
48     double factor_of_10 = pow(10,number_of_digits->numericVal);
49     return new_number(rint(num->numericVal * factor_of_10) / factor_of_10);
50 }
51
52 value_p extend_len(value_p str_val) {
53     if (!assertSingleString(str_val)) return new_val();
54     return new_number((double) str_val->str->length);
55 }
56
57 value_p extend_get_stdin() {
58     if (open_num_files + 1 > MAX_FILES) {
59         return new_val();
60     } else {
61         open_num_files++;
62         open_files[open_num_files] = stdin;
63         return new_number((double) open_num_files);
64     }
65 }
66
67 value_p extend_get_stdout() {
68     if (open_num_files + 1 > MAX_FILES) {
69         return new_val();
70     } else {
71         open_num_files++;
72         open_files[open_num_files] = stdout;
73         return new_number((double) open_num_files);
74     }
75 }
76
77 value_p extend_get_stderr() {
78     if (open_num_files + 1 > MAX_FILES) {
79         return new_val();
80     } else {
81         open_num_files++;
82         open_files[open_num_files] = stderr;
83         return new_number((double) open_num_files);
84     }
85 }
86
87 value_p extend_open(value_p filename, value_p mode){
88     FILE *val;
89     if ( !assertSingleString(filename)
90         || !assertSingleString(mode)
91         || open_num_files + 1 > MAX_FILES) {
92         return new_val();
93     }
94     val = fopen(filename->str->text, mode->str->text);
95     if(val == NULL) return new_val();
96     open_num_files++;
97     open_files[open_num_files] = val;
98     return new_number((double) open_num_files);
99 }
100
101 value_p extend_close(value_p file_handle) {
102     if(!assertSingleNumber(file_handle)) {

```

```

103     // Per the LRM this is actually supposed to crash the program.
104     fprintf(stderr, "EXITING - Attempted to close something that was not a valid file
        pointer\n");
105     exit(-1);
106 }
107 int fileNum = (int) file_handle->numericVal;
108
109 if (fileNum > open_num_files || open_files[fileNum] == NULL) {
110     // Per the LRM this is actually supposed to crash the program.
111     fprintf(stderr, "EXITING - Attempted to close something that was not a valid file
        pointer\n");
112     exit(-1);
113 }
114 fclose(open_files[fileNum]);
115 open_files[fileNum] = NULL; // Empty the container for the pointer.
116 return new_val(); // assuming it was an open valid handle, close() is just supposed
        to return empty
117 }
118
119 value_p extend_read(value_p file_handle, value_p num_bytes){
120     /* TODO: Make it accept empty */
121     if(!assertSingleNumber(file_handle) || !assertSingleNumber(num_bytes)) return
        new_val();
122     int max_bytes = (int)num_bytes->numericVal;
123     int fileNum = (int)file_handle->numericVal;
124     if (fileNum > open_num_files || open_files[fileNum] == NULL) return new_val();
125     FILE *f = open_files[fileNum];
126     max_bytes = (int) num_bytes->numericVal;
127     if (max_bytes == 0) {
128         long cur_pos = ftell(f);
129         fseek(f, 0, SEEK_END);
130         long end_pos = ftell(f);
131         fseek(f, cur_pos, SEEK_SET);
132         max_bytes = end_pos - cur_pos;
133     }
134     char *buf = malloc(sizeof(char) * (max_bytes + 1));
135     int bytes_read = fread(buf, sizeof(char), max_bytes, f);
136     buf[bytes_read] = 0;
137     value_p result = new_string(buf);
138     free(buf);
139     return result;
140     //edge case: how to return the entire contents of the file if n == empty?
141 }
142
143 value_p extend_readline(value_p file_handle) {
144     int i=0, buf_size = 256;
145     char next_char;
146     if (!assertSingleNumber(file_handle)) return new_val();
147     int fileNum = (int) file_handle->numericVal;
148     FILE *f = open_files[fileNum];
149     if (fileNum > open_num_files || open_files[fileNum] == NULL) {
150         return new_val();
151     }
152     char *buf = (char *) malloc (buf_size * sizeof(char));
153     while ((next_char = fgetc(f)) != '\n') {
154         buf[i++] = next_char;

```

```

155     if (i == buf_size - 2) {
156         buf_size *= 2;
157         char *new_buf = (char *) malloc (buf_size * sizeof(char));
158         memcpy(new_buf, buf, i);
159         free(buf);
160         buf = new_buf;
161     }
162 }
163 buf[i] = '\0';
164 value_p result = new_string(buf);
165 free(buf);
166 return result;
167 }
168
169 value_p extend_write(value_p file_handle, value_p buffer){
170     if(!assertSingleNumber(file_handle) || !assertSingleString(buffer)) return new_val()
171     ;
172     int fileNum = (int) file_handle->numericVal;
173     if (fileNum > open_num_files || open_files[fileNum] == NULL) {
174         // Per the LRM this is actually supposed to crash the program.
175         fprintf(stderr, "EXITING - Attempted to write to something that was not a valid
176             file pointer\n");
177         exit(-1);
178     }
179     fwrite(buffer->str->text, 1, buffer->str->length, open_files[fileNum]);
180     // TODO: make this return empty once compiler handles Id(s)
181     // RN: Use the return value to close the file
182     return new_number((double) fileNum);
183 }
184
185 #ifdef PLOT
186 value_p extend_plot(value_p file_name){
187     // extract the numerical values from the first parameter - values
188     if(!assertSingle(file_name)) return new_val();
189     float a[6] = { 0.5, 0.09, 0.6, 0.85, 0.0, 0.90 },
190           b[6] = { 1.9, 1.3, 0.6, 0.75, 0.1, 2.0 };
191     char *t[6] = { "Chicago", "New York", "L.A.", "Atlanta", "Paris, MD\n(USA) ", "
192         London" };
193     unsigned long sc[2] = { 0xFF8080, 0x8080FF };
194     GDC_BGColor = 0xFFFFFFFFL;
195     GDC_LineColor = 0x000000L;
196     GDC_SetColor = &(sc[0]);
197     GDC_stack_type = GDC_STACK_BESIDE;
198     // Using the line below, can also spit to stdout and fwrite from Extend
199     // printf( "Content-Type: image/png\n\n" );
200     FILE *outpng = fopen("extend.png", "wb");
201     out_graph(250, 200, outpng, GDC_3DBAR, 6, t, 2, a, b);
202     fclose(outpng);
203     return new_val();
204 }
205
206 value_p extend_bar_chart(value_p file_handle, value_p labels, value_p values){
207     // Mandates 1 row, X columns
208     if(!assertSingleNumber(file_handle)) return new_val();
209     int fileNum = (int)file_handle->numericVal;
210     if (fileNum > open_num_files || open_files[fileNum] == NULL) return new_val();

```



```

208 FILE *f = open_files[fileNum];
209 int data_length = labels->subrange->subrange_num_cols;
210 if(data_length != values->subrange->subrange_num_cols) return new_val();
211
212 float *graph_values = malloc(sizeof(float) * data_length);
213 char **graph_labels = malloc(sizeof(char*) * data_length);
214 for(int i = 0; i < data_length; i++){
215     graph_labels[i] = getValSR(labels->subrange, 0, i)->str->text;
216     graph_values[i] = (float)getValSR(values->subrange, 0, i)->numericVal;
217 }
218 unsigned long sc[2] = {0xFF8080, 0x8080FF};
219 GDC_BGColor = 0xFFFFFFL;
220 GDC_LineColor = 0x000000L;
221 GDC_SetColor = &(sc[0]);
222 GDC_stack_type = GDC_STACK_BESIDE;
223 out_graph(250, 200, f, GDC_3DBAR, data_length, graph_labels, 1, graph_values);
224 // width, height, file handle, graph type, number of data points, labels, number of
    data sets, the data sets
225 free(graph_labels);
226 free(graph_values);
227 fclose(f);
228 return new_val();
229 }
230
231 value_p extend_line_chart(value_p file_handle, value_p labels, value_p x_values){
232     if(!assertSingleNumber(file_handle)) return new_val();
233     int fileNum = (int)file_handle->numericVal;
234     if (fileNum > open_num_files || open_files[fileNum] == NULL) return new_val();
235     FILE *f = open_files[fileNum];
236     int data_length = labels->subrange->subrange_num_cols;
237     if(data_length != x_values->subrange->subrange_num_cols) return new_val();
238     float *graph_x_values = malloc(sizeof(float) * data_length);
239     char **graph_labels = malloc(sizeof(char*) * data_length);
240     for(int i = 0; i < data_length; i++){
241         graph_labels[i] = getValSR(labels->subrange, 0, i)->str->text;
242         graph_x_values[i] = (float)getValSR(x_values->subrange, 0, i)->numericVal;
243     }
244     unsigned long sc[2] = {0xFF8080, 0x8080FF};
245     GDC_BGColor = 0xFFFFFFL;
246     GDC_LineColor = 0x000000L;
247     GDC_SetColor = &(sc[0]);
248     GDC_stack_type = GDC_STACK_BESIDE;
249     out_graph(250, 200, f, GDC_LINE, data_length, graph_labels, 1, graph_x_values);
250     free(graph_labels);
251     free(graph_x_values);
252     fclose(f);
253     return new_val();
254 }
255 #endif
256
257 value_p extend_isNaN(value_p val) {
258     if (!assertSingleNumber(val)) return new_val();
259     double d = val->numericVal;
260     return isnan(d) ? new_number(1.0) : new_number(0.0);
261 }
262

```

```

263 value_p extend_isInfinite(value_p val) {
264     if (!assertSingleNumber(val)) return new_val();
265     double d = val->numericVal;
266     if (isinf(d)) {
267         return d < 0 ? new_number(-1.0) : new_number(1.0);
268     } else {
269         return new_number(0.0);
270     }
271 }
272
273 value_p extend_parseFloat(value_p val) {
274     if (!assertSingleString(val)) return new_val();
275     return new_number(atof(val->str->text));
276 }
277
278 value_p extend_toASCII(value_p val) {
279     if (!assertSingleString(val) || val->str->length == 0) return new_val();
280     value_p *val_arr = malloc(sizeof(value_p) * val->str->length);
281     int i;
282     for(i = 0; i < val->str->length; i++) {
283         value_p my_val = malloc(sizeof(struct value_t));
284         my_val->flags = FLAG_NUMBER;
285         my_val->numericVal = (double)val->str->text[i];
286         val_arr[i] = my_val;
287     }
288     value_p _new = new_subrange(1, val->str->length, val_arr);
289     return _new;
290 }
291
292 value_p extend_fromASCII(value_p val) {
293     if(val->flags == FLAG_NUMBER) {
294         char s[2];
295         s[0] = ((char)lrint(val->numericVal));
296         s[1] = '\0';
297         return new_string(s);
298     }
299     else if(val->flags == FLAG_SUBRANGE) {
300         int rows, cols, len;
301         rows = val->subrange->subrange_num_rows;
302         cols = val->subrange->subrange_num_cols;
303         if(rows > 1 && cols > 1) return new_val();
304         else len = (rows == 1 ? cols : rows);
305         char *text = malloc(1 + sizeof(char) * len);
306         for(rows = 0; rows < val->subrange->subrange_num_rows; rows++) {
307             for(cols = 0; cols < val->subrange->subrange_num_cols; cols++) {
308                 value_p single = getValSR(val->subrange, rows, cols);
309                 if(single->flags != FLAG_NUMBER) {
310                     free(text);
311                     return new_val();
312                 }
313                 text[rows + cols] = (char)lrint(single->numericVal);
314             }
315         }
316         text[len] = '\0';
317         value_p ret = new_string(text);
318         free(text);

```

```

319     return ret;
320 } else if (val->flags == FLAG_EMPTY) {
321     return new_string("");
322 } else {
323     return new_val();
324 }
325 }

```

## 9.11 runtime.c

```

1  /* jss2272 ns3158 */
2
3  #include<stdio.h>
4  #include<stdlib.h>
5  #include<math.h>
6  #include<sys/resource.h>
7  #include<string.h>
8  #include<stdbool.h>
9  #include "runtime.h"
10
11 struct value_t zero_val = {FLAG_NUMBER, 0.0, NULL, NULL};
12 struct value_t one_val = {FLAG_NUMBER, 1.0, NULL, NULL};
13 struct rhs_index absolute_zero = {&zero_val, RHS_IDX_ABSOLUTE};
14 struct rhs_index absolute_one = {&one_val, RHS_IDX_ABSOLUTE};
15 struct rhs_slice zero_to_one = {&absolute_zero, &absolute_one};
16 struct rhs_slice corresponding_cell = {NULL, NULL};
17
18 void debug_print_subrange(subrange_p subrng);
19
20 void debug_print(value_p val, char *which_value) {
21     char *flag_meanings[4] = {"Empty", "Number", "String", "Subrange"};
22     fprintf(stderr, "-----Everything you ever wanted to know about %s:-----\n",
23             which_value == NULL ? "some anonymous variable" : which_value);
24     fprintf(stderr, "Memory address: %p\n", val);
25     if (val == NULL) {
26         fprintf(stderr, "-----Nice try asking me to dereference a null pointer\n
27             -----");
28         return;
29     }
30     fprintf(stderr, "Flags: %d (%s)\n", val->flags, flag_meanings[val->flags]);
31     fprintf(stderr, "NumericVal: %f\n", val->numericVal);
32     fprintf(stderr, "String contents: Probably safer not to check that pointer (%p)
33         blindly\n", val->str);
34     if (val->flags == FLAG_STRING && val->str != NULL) {
35         fprintf(stderr, "It says it's a string and it's not a NULL pointer though, so here
36             you go:\n");
37         fprintf(stderr, "String refcount: %d\n", val->str->refs);
38         fprintf(stderr, "String length: %ld\n", val->str->length);
39         fprintf(stderr, "String char* memory address: %p\n", val->str->text);
40         if (val->str->text == NULL) {
41             fprintf(stderr, "Not going to print the contents of NULL!\n");
42         } else {
43             fprintf(stderr, "String char* contents:\n%s\n", val->str->text);
44         }
45     }
46 }

```

```

42     fprintf(stderr, "Subrange contents: Probably safer not to check that pointer (%p)
        blindly either\n", val->subrange);
43     if (val->flags == FLAG_SUBRANGE && val->subrange != NULL) {
44         fprintf(stderr, "It says it's a subrange and it's not a NULL pointer though, so
            here you go:\n");
45         debug_print_subrange(val->subrange);
46     }
47     fprintf(stderr, "-----That's all I've got to say about %s:-----\n", which_value ==
        NULL ? "some anonymous variable" : which_value);
48 }
49
50 void debug_print_formula(struct ExtendFormula *fdef) {
51     fprintf(stderr, "-----Everything you ever wanted to know about your favorite
        formula:-----\n");
52     fprintf(stderr, "RowStart varnum: %d %d\n", fdef->rowStart_varnum, fdef->
        fromFirstRow);
53     fprintf(stderr, "RowEnd varnum: %d %d\n", fdef->rowEnd_varnum, fdef->toLastRow);
54     fprintf(stderr, "ColStart varnum: %d %d\n", fdef->colStart_varnum, fdef->
        fromFirstCol);
55     fprintf(stderr, "ColEnd varnum: %d %d\n", fdef->colEnd_varnum, fdef->toLastCol);
56 }
57
58 void debug_print_res_formula(struct ResolvedFormula *rdef) {
59     fprintf(stderr, "Some formula with function pointer %p applies to: [%d:%d,%d:%d]\n",
        rdef->formula, rdef->rowStart, rdef->rowEnd, rdef->colStart, rdef->colEnd);
60 }
61
62 void debug_print_vardefn(struct var_defn *pdef) {
63     fprintf(stderr, "-----Everything you ever wanted to know about var defn %s:-----\n
        ", pdef->name);
64     fprintf(stderr, "Row varnum: %d\n", pdef->rows_varnum);
65     fprintf(stderr, "Col varnum: %d\n", pdef->cols_varnum);
66     fprintf(stderr, "Num formulas: %d\n", pdef->numFormulas);
67     fprintf(stderr, "Formula defs: \n");
68     int i;
69     for (i=0; i < pdef->numFormulas; i++) {
70         debug_print_formula(pdef->formulas + i);
71     }
72     fprintf(stderr, "Is 1x1: %d\n", pdef->isOneByOne);
73 }
74
75 void debug_print_varinst(struct var_instance *inst) {
76     fprintf(stderr, "-----Everything you ever wanted to know about var %s:-----\n",
        inst->name);
77     fprintf(stderr, "Rows: %d\n", inst->rows);
78     fprintf(stderr, "Cols: %d\n", inst->cols);
79     fprintf(stderr, "Num formulas: %d\n", inst->numFormulas);
80     fprintf(stderr, "*****Formulas:*****\n");
81     int i;
82     for (i = 0; i < inst->numFormulas; i++) {
83         debug_print_res_formula(inst->formulas + i);
84     }
85     fprintf(stderr, "**** End of Formulas *** \n");
86     fprintf(stderr, "~~~~~~~Cells:~~~~~~~\n");
87     fprintf(stderr, "Status memory address: %p\n", inst->status);
88     for (i = 0; i < inst->rows * inst->cols; i++) {

```

```

89     printf("%s[%d,%d]: Status=%d\n", inst->name, i / inst->cols, i % inst->cols, inst
        ->status[i]);
90     if (inst->status[i] == CALCULATED) {
91         printf("%s[%d,%d] Value:\n", inst->name, i / inst->cols, i % inst->cols);
92         debug_print(inst->values[i], inst->name);
93     }
94 }
95 fprintf(stderr, "~~~ End of Cells: ~~~\n");
96 }
97
98 void debug_print_subrange(subrange_p subrng) {
99     fprintf(stderr, "~~~~~Everything you wanted to know about this subrange~~~~~\n");
100    fprintf(stderr, "Offset: [%d,%d]\n", subrng->base_var_offset_row, subrng->
        base_var_offset_col);
101    fprintf(stderr, "Dimensions: [%d,%d]\n", subrng->subrange_num_rows, subrng->
        subrange_num_cols);
102    fprintf(stderr, "Subrange of: \n");
103    debug_print_varinst(subrng->range);
104 }
105
106 void debug_print_index(struct rhs_index *idx) {
107     if (idx == NULL) {
108         fprintf(stderr, "I'd rather not try to print out the contents of a NULL index.\n");
109         ;
110         exit(-1);
111     }
112     fprintf(stderr, "Index type: ");
113     switch(idx->rhs_index_type) {
114     case RHS_IDX_ABSOLUTE:
115         fprintf(stderr, "Absolute\n");
116         if (idx->val_of_expr == NULL) {
117             fprintf(stderr, "I wasn't expecting this, but the value pointer is NULL. Maybe
                there's a good reason for it, so I'll keep going...\n");
118         } else {
119             debug_print(idx->val_of_expr, "an absolute index");
120         }
121         break;
122     case RHS_IDX_RELATIVE:
123         fprintf(stderr, "Relative\n");
124         if (idx->val_of_expr == NULL) {
125             fprintf(stderr, "I wasn't expecting this, but the value pointer is NULL. Maybe
                there's a good reason for it, so I'll keep going...\n");
126         } else {
127             debug_print(idx->val_of_expr, "a relative index");
128         }
129         break;
130     case RHS_IDX_DIM_START:
131         fprintf(stderr, "DimensionStart\n");
132         if (idx->val_of_expr != NULL) {
133             fprintf(stderr, "This definitely isn't supposed to happen - the value pointer
                isn't NULL. You should look into that.\n");
134             exit(-1);
135         }
136         break;
137     case RHS_IDX_DIM_END:
138         fprintf(stderr, "DimensionEnd\n");

```

```

138     if (idx->val_of_expr != NULL) {
139         fprintf(stderr, "This definitely isn't supposed to happen - the value pointer
            isn't NULL. You should look into that.\n");
140         exit(-1);
141     }
142     break;
143 }
144 }
145
146 void debug_print_slice(struct rhs_slice *sl) {
147     if (sl == NULL) {
148         fprintf(stderr, "I'd rather not try to print out the contents of a NULL slice.\n")
            ;
149         exit(-1);
150     }
151     fprintf(stderr, "-----Everything about this slice-----\n");
152     fprintf(stderr, "Start and end index memory addresses: %p and %p\n", sl->
        slice_start_index, sl->slice_end_index);
153     if (sl->slice_start_index != NULL) {
154         fprintf(stderr, "Start index info:\n");
155         debug_print_index(sl->slice_start_index);
156         if (sl->slice_end_index != NULL) {
157             fprintf(stderr, "End index info:\n");
158             debug_print_index(sl->slice_end_index);
159         }
160     } else {
161         if (sl->slice_end_index != NULL) {
162             fprintf(stderr, "Start index is NULL but end index is not NULL. That should
                never happen.\n");
163             fprintf(stderr, "Attempting to print contents anyway:\n");
164             fflush(stderr);
165             debug_print_index(sl->slice_end_index);
166         }
167     }
168 }
169
170 void debug_print_selection(struct rhs_selection *sel) {
171     if (sel == NULL) {
172         fprintf(stderr, "I'd rather not try to print out the contents of a NULL selection
            .\n");
173         exit(-1);
174     }
175     fprintf(stderr, "-----Everything about this selection-----\n");
176     fprintf(stderr, "Slice memory addresses: %p and %p\n", sel->slicel, sel->slice2);
177     if (sel->slicel != NULL) {
178         fprintf(stderr, "Slice 1 info:\n");
179         debug_print_slice(sel->slicel);
180         if (sel->slice2 != NULL) {
181             fprintf(stderr, "Slice 2 info:\n");
182             debug_print_slice(sel->slice2);
183         }
184     } else {
185         if (sel->slice2 != NULL) {
186             fprintf(stderr, "Slice 1 is NULL but slice 2 is not NULL. That should never
                happen.\n");
187             fprintf(stderr, "Attempting to print contents anyway:\n");

```

```

188     fflush(stderr);
189     debug_print_slice(sel->slice2);
190 }
191 }
192 fprintf(stderr, "————That's all I've got about that selection————\n\n");
193 }
194
195 int rg_eq(value_p val1, value_p val2) {
196     int res = 1;
197     if(val1->flags != val2->flags) res = 0;
198     else if(val1->flags == FLAG_EMPTY) ;
199     else if(val1->flags == FLAG_NUMBER && val1->numericVal != val2->numericVal) res = 0;
200     else if(val1->flags == FLAG_STRING && strcmp(val1->str->text, val2->str->text)) res
        = 0;
201     else if(val1->flags == FLAG_SUBRANGE) {
202         subrange_p sr1 = val1->subrange;
203         subrange_p sr2 = val2->subrange;
204         if(sr1->subrange_num_cols != sr2->subrange_num_cols || sr1->subrange_num_rows !=
            sr2->subrange_num_rows) {
205             return 0;
206         } else {
207             int i, j;
208             value_p v1, v2;
209             for(i = 0; i < sr1->subrange_num_rows; i++) {
210                 for(j = 0; j < sr1->subrange_num_cols; j++) {
211                     v1 = getValSR(sr1, i, j);
212                     v2 = getValSR(sr2, i, j);
213                     if(rg_eq(v1, v2) == 0) {
214                         return 0;
215                     }
216                 }
217             }
218         }
219     }
220     return res;
221 }
222
223 void incStack() {
224     const rlim_t kStackSize = 64L * 1024L * 1024L;
225     struct rlimit rl;
226     int result;
227
228     result = getrlimit(RLIMIT_STACK, &rl);
229     rl.rlim_cur = rl.rlim_max;
230     result = setrlimit(RLIMIT_STACK, &rl);
231 }
232
233 double setNumeric(value_p result, double val) {
234     result->flags = FLAG_NUMBER;
235     return (result->numericVal = val);
236 }
237
238 double setFlag(value_p result, double flag_num) {
239     return (result->flags = flag_num);
240 }
241

```

```

242 int assertSingle(value_p value) {
243     /* TODO: dereference 1 by 1 subrange */
244     return !(value->flags == FLAG_SUBRANGE);
245 }
246
247 int assertSingleNumber(value_p p) {
248     if (!assertSingle(p)) {
249         return 0;
250     }
251     return (p->flags == FLAG_NUMBER);
252 }
253
254 int assertText(value_p my_val) {
255     return (my_val->flags == FLAG_STRING);
256 }
257
258 int assertSingleString(value_p p) {
259     if (!assertSingle(p)) {
260         return 0;
261     }
262     return (p->flags == FLAG_STRING);
263 }
264
265 int assertEmpty(value_p p) {
266     if (!assertSingle(p)) {
267         return 0;
268     }
269     return (p->flags == FLAG_EMPTY);
270 }
271
272 value_p new_val() {
273     value_p empty_val = malloc(sizeof(struct value_t));
274     setFlag(empty_val, FLAG_EMPTY);
275     return empty_val;
276 }
277
278 value_p new_number(double val) {
279     value_p new_v = malloc(sizeof(struct value_t));
280     setFlag(new_v, FLAG_NUMBER);
281     setNumeric(new_v, val);
282     return new_v;
283 }
284
285 value_p new_string(char *s) {
286     if (s == NULL) return new_val();
287     value_p new_v = malloc(sizeof(struct value_t));
288     setFlag(new_v, FLAG_STRING);
289     string_p new_str = malloc(sizeof(struct string_t));
290     long len = strlen(s);
291     new_str->text = malloc(len+1);
292     strcpy(new_str->text, s);
293     new_str->length = len;
294     new_str->refs = 1;
295     new_v->str = new_str;
296     return new_v;
297 }

```



```

298
299 struct ExtendScope *global_scope;
300
301 void null_init(struct ExtendScope *scope_ptr) {
302     int i;
303     for(i = 0; i < scope_ptr->numVars; i++)
304         scope_ptr->vars[i] = NULL;
305 }
306
307 char getIntFromOneByOne(struct ExtendScope *scope_ptr, int varnum, int *result) {
308     if (!scope_ptr->defns[varnum].isOneByOne) {
309         fprintf(stderr, "A variable (%s) that is supposedly one by one is not defined that
310             way.\n", scope_ptr->defns[varnum].name);
311         exit(-1);
312     }
313     struct var_instance *inst = get_variable(scope_ptr, varnum);
314     if (inst->rows != 1 || inst->cols != 1) {
315         fprintf(stderr, "A variable (%s) that is defined as one by one is somehow actually
316             %d by %d.\n", inst->name, inst->rows, inst->cols);
317         exit(-1);
318     }
319     value_p val = getVal(inst, 0, 0);
320     if (!assertSingleNumber(val) || !isfinite(val->numericVal)) {
321         return 0;
322     }
323     *result = (int) lrint(val->numericVal);
324     return 1;
325 }
326
327 struct var_instance *instantiate_variable(struct ExtendScope *scope_ptr, struct
328     var_defn def) {
329     struct var_instance *inst = malloc(sizeof(struct var_instance));
330     if(def.isOneByOne) {
331         inst->rows = 1;
332         inst->cols = 1;
333     } else {
334         if (!getIntFromOneByOne(scope_ptr, def.rows_varnum, &inst->rows)) {
335             fprintf(stderr, "EXITING - The expression for the number of rows of variable %s
336                 did not evaluate to a finite Number.\n", def.name);
337             exit(-1);
338         }
339         if (!getIntFromOneByOne(scope_ptr, def.cols_varnum, &inst->cols)) {
340             fprintf(stderr, "EXITING - The expression for the number of columns of variable
341                 %s did not evaluate to a finite Number.\n", def.name);
342             exit(-1);
343         }
344         if (inst->rows <= 0 || inst->cols <= 0) {
345             fprintf(stderr, "EXITING - The requested dimensions for variable %s were [%d, %d
346                 ]; they must both be greater than zero.\n", def.name, inst->rows, inst->cols)
347                 ;
348             exit(-1);
349         }
350     }
351     // TODO: do the same thing for each FormulaFP to turn an ExtendFormula into a
352     ResolvedFormula
353     inst->numFormulas = def.numFormulas;

```

```

346 inst->closure = scope_ptr;
347 inst->name = def.name;
348 int size = inst->rows * inst->cols;
349 inst->values = malloc(sizeof(value_p) * size);
350 memset(inst->values, 0, sizeof(value_p) * size);
351 inst->status = malloc(sizeof(char) * size);
352 memset(inst->status, 0, sizeof(char) * size);
353 inst->formulas = malloc(sizeof(struct ResolvedFormula) * inst->numFormulas);
354 //debug_print_vardefn(&def);
355 //debug_print_varinst(inst);
356 int i, j;
357 for(i = 0; i < inst->numFormulas; i++) {
358
359     // Set the formula function pointer to the pointer from the definition
360     inst->formulas[i].formula = def.formulas[i].formula;
361
362     if (def.isOneByOne) {
363         inst->formulas[i].rowStart = 0;
364         inst->formulas[i].rowEnd = 1;
365         inst->formulas[i].colStart = 0;
366         inst->formulas[i].colEnd = 1;
367     } else {
368         if(def.formulas[i].fromFirstRow) {
369             inst->formulas[i].rowStart = 0;
370         } else {
371             if (!getIntFromOneByOne(scope_ptr, def.formulas[i].rowStart_varnum, &inst->
                 formulas[i].rowStart)) {
372                 fprintf(stderr, "EXITING - The requested starting row for formula %d of %s
                 did not evaluate to a finite number.\n", i, inst->name);
373                 exit(-1);
374             }
375             if (inst->formulas[i].rowStart < 0) {
376                 inst->formulas[i].rowStart += inst->rows;
377             }
378             if (inst->formulas[i].rowStart < 0 || inst->formulas[i].rowStart >= inst->rows
                 ) {
379                 //Doesn't matter, but will never get called
380             }
381         }
382         if (def.formulas[i].isSingleRow) {
383             inst->formulas[i].rowEnd = inst->formulas[i].rowStart + 1;
384         } else if (def.formulas[i].toLastRow) {
385             inst->formulas[i].rowEnd = inst->rows;
386         } else {
387             if (!getIntFromOneByOne(scope_ptr, def.formulas[i].rowEnd_varnum, &inst->
                 formulas[i].rowEnd)) {
388                 fprintf(stderr, "EXITING - The requested ending row for formula %d of %s did
                 not evaluate to a finite number.\n", i, inst->name);
389                 exit(-1);
390             }
391             if (inst->formulas[i].rowEnd < 0) {
392                 inst->formulas[i].rowEnd += inst->rows;
393             }
394         }
395         if(def.formulas[i].fromFirstCol) {
396             inst->formulas[i].colStart = 0;

```

```

397     } else {
398         if (!getIntFromOneByOne(scope_ptr, def.formulas[i].colStart_varnum, &inst->
            formulas[i].colStart)) {
399             fprintf(stderr, "EXITING - The requested starting column for formula %d of %
                s did not evaluate to a finite number.\n", i, inst->name);
400             exit(-1);
401         }
402         if (inst->formulas[i].colStart < 0) {
403             inst->formulas[i].colStart += inst->cols;
404         }
405         if (inst->formulas[i].colStart < 0 || inst->formulas[i].colStart >= inst->cols
            ) {
406             //Doesn't matter, but will never get called
407         }
408     }
409     if (def.formulas[i].isSingleCol) {
410         inst->formulas[i].colEnd = inst->formulas[i].colStart + 1;
411     } else if (def.formulas[i].toLastCol) {
412         inst->formulas[i].colEnd = inst->cols;
413     } else {
414         if (!getIntFromOneByOne(scope_ptr, def.formulas[i].colEnd_varnum, &inst->
            formulas[i].colEnd)) {
415             fprintf(stderr, "EXITING - The requested starting column for formula %d of %
                s did not evaluate to a finite number.\n", i, inst->name);
416             exit(-1);
417         }
418         if (inst->formulas[i].colEnd < 0) {
419             inst->formulas[i].colEnd += inst->cols;
420         }
421     }
422 }
423 }
424
425 for (i = 1; i < inst->numFormulas; i++) {
426     for (j = 0; j < i; j++) {
427         int intersectRowStart = (inst->formulas[i].rowStart > inst->formulas[j].rowStart
            ) ? inst->formulas[i].rowStart : inst->formulas[j].rowStart;
428         int intersectColStart = (inst->formulas[i].colStart > inst->formulas[j].colStart
            ) ? inst->formulas[i].colStart : inst->formulas[j].colStart;
429         int intersectRowEnd = (inst->formulas[i].rowEnd < inst->formulas[j].rowEnd) ?
            inst->formulas[i].rowEnd : inst->formulas[j].rowEnd;
430         int intersectColEnd = (inst->formulas[i].colEnd < inst->formulas[j].colEnd) ?
            inst->formulas[i].colEnd : inst->formulas[j].colEnd;
431         if (intersectRowEnd > intersectRowStart && intersectColEnd > intersectColStart)
            {
432             fprintf(stderr, "Runtime error: Multiple formulas were assigned to %s[%d:%d,%d
                :%d].\n", inst->name,
433                 intersectRowStart, intersectRowEnd, intersectColStart,
                    intersectColEnd);
434             exit(-1);
435         }
436     }
437 }
438
439 scope_ptr->refcount++;
440 return inst;

```

```

441 }
442
443 struct var_instance *get_variable(struct ExtendScope *scope_ptr, int varnum) {
444     if (varnum >= scope_ptr->numVars) {
445         fprintf(stderr, "Runtime error: Asked for nonexistant variable number\n");
446         exit(-1);
447     }
448     if (scope_ptr->vars[varnum] == NULL) {
449         scope_ptr->vars[varnum] = instantiate_variable(scope_ptr, scope_ptr->defns[varnum]
450     );
451     }
452     return scope_ptr->vars[varnum];
453 }
454
455 char assertInBounds(struct var_instance *defn, int r, int c) {
456     return (
457         r >= 0 && r < defn->rows &&
458         c >= 0 && c < defn->cols
459     );
460 }
461
462 value_p calcVal(struct var_instance *inst, int r, int c) {
463     int i;
464     for (i = 0; i < inst->numFormulas; i++) {
465         if (
466             r >= inst->formulas[i].rowStart && r < inst->formulas[i].rowEnd &&
467             c >= inst->formulas[i].colStart && c < inst->formulas[i].colEnd
468         ) {
469             return (inst->formulas[i].formula)(inst->closure, r, c);
470         }
471     }
472     return new_val();
473 }
474
475 value_p clone_value(value_p old_value) {
476     value_p new_value = (value_p) malloc(sizeof(struct value_t));
477     new_value->flags = old_value->flags;
478     switch (new_value->flags) {
479         case FLAG_EMPTY:
480             break;
481         case FLAG_NUMBER:
482             new_value->numericVal = old_value->numericVal;
483             break;
484         case FLAG_STRING:
485             new_value->str = old_value->str;
486             new_value->str->refs++;
487             break;
488         case FLAG_SUBRANGE:
489             new_value->subrange = (subrange_p) malloc(sizeof(struct subrange_t));
490             memcpy(new_value->subrange, old_value->subrange, sizeof(struct subrange_t));
491             if (new_value->subrange->range->closure != NULL) {
492                 new_value->subrange->range->closure->refcount++; /* Not sure about this one */
493             }
494             break;
495         default:
496             fprintf(stderr, "clone_value(%p): Illegal value of flags: %c\n", old_value,

```

```

        new_value->flags);
496     exit(-1);
497     break;
498 }
499 return new_value;
500 }
501
502 void delete_string_p(string_p old_string) {
503     old_string->refs--;
504     if (old_string->refs == 0) {
505         /* free(old_string); */
506     }
507 }
508
509 void delete_subrange_p(subrange_p old_subrange) {
510     if (old_subrange->range->closure != NULL) {
511         old_subrange->range->closure->refcount--;
512     }
513     free(old_subrange);
514 }
515
516 void delete_value(value_p old_value) {
517     switch (old_value->flags) {
518         case FLAG_EMPTY:
519             break;
520         case FLAG_NUMBER:
521             break;
522         case FLAG_STRING:
523             delete_string_p(old_value->str); /* doesn't do anything besides decrement the
524                 ref count now */
525             break;
526         case FLAG_SUBRANGE:
527             delete_subrange_p(old_value->subrange);
528             break;
529         default:
530             fprintf(stderr, "delete_value(%p): Illegal value of flags: %c\n", old_value,
531                 old_value->flags);
532             exit(-1);
533             break;
534     }
535 }
536
537 value_p deref_subrange_p(subrange_p subrng) {
538     if (subrng == NULL) {
539         fprintf(stderr, "Exiting - asked to dereference a NULL pointer.\n");
540         exit(-1);
541     }
542     if (subrng->subrange_num_rows == 1 && subrng->subrange_num_cols == 1) {
543         return getVal(subrng->range, subrng->base_var_offset_row, subrng->
544             base_var_offset_col);
545     } else {
546         value_p new_value = (value_p) malloc (sizeof(struct value_t));
547         new_value->flags = FLAG_SUBRANGE;
548         new_value->numericVal = 0.0;
549         new_value->str = NULL;
550         new_value->subrange = (subrange_p) malloc (sizeof(struct subrange_t));

```

```

548     memcpy(new_value->subrange, subrng, sizeof(struct subrange_t));
549     if (new_value->subrange->range->closure != NULL) {
550         new_value->subrange->range->closure->refcount++;
551     }
552     return new_value;
553 }
554 }
555
556 value_p new_subrange(int num_rows, int num_cols, value_p *vals) {
557     /* This function does not check its arguments; if you supply fewer
558      * than num_rows * num_cols elements in vals, it will crash.
559      * Only use this function if you know what you're doing. */
560     struct subrange_t sr;
561     sr.range = (struct var_instance *) malloc (sizeof(struct var_instance));
562     sr.base_var_offset_row = 0;
563     sr.base_var_offset_col = 0;
564     sr.subrange_num_rows = num_rows;
565     sr.subrange_num_cols = num_cols;
566     sr.range->rows = num_rows;
567     sr.range->cols = num_cols;
568     sr.range->numFormulas = 0;
569     sr.range->formulas = NULL;
570     sr.range->closure = NULL;
571     sr.range->values = (value_p *) malloc(num_rows * num_cols * sizeof(value_p));
572     sr.range->status = (char *) malloc (num_rows * num_cols * sizeof(char));
573     sr.range->name = NULL;
574     int i;
575     for (i = 0; i < num_rows * num_cols; i++) {
576         sr.range->values[i] = clone_value(vals[i]);
577         sr.range->status[i] = CALCULATED;
578     }
579     return deref_subrange_p(&sr);
580 }
581
582 value_p box_command_line_args(int argc, char **argv) {
583     value_p *vals = (value_p *) malloc (argc * sizeof(value_p));
584     int i;
585     for (i = 0; i < argc; i++) {
586         vals[i] = new_string(argv[i]);
587     }
588     value_p ret = new_subrange(1, argc, vals);
589     for (i = 0; i < argc; i++) {
590         free(vals[i]);
591     }
592     free(vals);
593     return ret;
594 }
595
596 char resolve_rhs_index(struct rhs_index *index, int dimension_len, int
    dimension_cell_num, int *result_ptr) {
597     if (index == NULL) {
598         fprintf(stderr, "Exiting - asked to dereference a NULL index\n");
599         exit(-1);
600     }
601     int i;
602     switch(index->rhs_index_type) {

```

```

603     case RHS_IDX_ABSOLUTE:
604         if (!assertSingleNumber(index->val_of_expr)) return false;
605         i = (int) lrint(index->val_of_expr->numericVal);
606         if (i >= 0) {
607             *result_ptr = i;
608         } else {
609             *result_ptr = i + dimension_len;
610         }
611         return true;
612         break;
613     case RHS_IDX_RELATIVE:
614         if (!assertSingleNumber(index->val_of_expr)) return false;
615         *result_ptr = dimension_cell_num + (int) lrint(index->val_of_expr->numericVal);
616         return true;
617         break;
618     case RHS_IDX_DIM_START:
619         *result_ptr = 0;
620         return true;
621         break;
622     case RHS_IDX_DIM_END:
623         *result_ptr = dimension_len;
624         return true;
625         break;
626     default:
627         fprintf(stderr, "Exiting - illegal index type\n");
628         exit(-1);
629         break;
630 }
631 }
632
633 char resolve_rhs_slice(struct rhs_slice *slice, int dimension_len, int
        dimension_cell_num, int *start_ptr, int *end_ptr) {
634     char start_success, end_success;
635     if (slice == NULL) {
636         fprintf(stderr, "Exiting - asked to dereference a NULL slice\n");
637         exit(-1);
638     }
639     if (slice->slice_start_index == NULL) {
640         if (slice->slice_end_index != NULL) {
641             fprintf(stderr, "Exiting - illegal slice\n");
642             exit(-1);
643         }
644         if (dimension_len == 1) {
645             *start_ptr = 0;
646             *end_ptr = 1;
647             return true;
648         } else {
649             *start_ptr = dimension_cell_num;
650             *end_ptr = dimension_cell_num + 1;
651             return true;
652         }
653     } else {
654         start_success = resolve_rhs_index(slice->slice_start_index, dimension_len,
            dimension_cell_num, start_ptr);
655         if (!start_success) return false;
656         if (slice->slice_end_index == NULL) {

```

```

657     *end_ptr = *start_ptr + 1;
658     return true;
659 } else {
660     end_success = resolve_rhs_index(slice->slice_end_index, dimension_len,
        dimension_cell_num, end_ptr);
661     return end_success;
662 }
663 }
664 }
665
666 value_p extract_selection(value_p expr, struct rhs_selection *sel, int r, int c) {
667     int expr_rows, expr_cols;
668     struct subrange_t subrange;
669     struct rhs_slice *row_slice_p, *col_slice_p;
670     int row_start, row_end, col_start, col_end;
671     char row_slice_success, col_slice_success;
672
673     if (expr == NULL || sel == NULL) {
674         fprintf(stderr, "Exiting - asked to extract a selection using a NULL pointer.\n");
675         exit(-1);
676     }
677     switch(expr->flags) {
678         case FLAG_EMPTY:
679             return new_val();
680             break;
681         case FLAG_NUMBER: case FLAG_STRING:
682             expr_rows = 1;
683             expr_cols = 1;
684             break;
685         case FLAG_SUBRANGE:
686             expr_rows = expr->subrange->subrange_num_rows;
687             expr_cols = expr->subrange->subrange_num_cols;
688             break;
689         default:
690             fprintf(stderr, "Exiting - invalid value type\n");
691             exit(-1);
692             break;
693     }
694     if (sel->slice1 == NULL) {
695         if (sel->slice2 != NULL) {
696             fprintf(stderr, "Exiting - illegal selection\n");
697             exit(-1);
698         }
699         row_slice_p = &corresponding_cell;
700         col_slice_p = &corresponding_cell;
701     } else {
702         if (sel->slice2 == NULL) {
703             if (expr_rows == 1) {
704                 row_slice_p = &zero_to_one;
705                 col_slice_p = sel->slice1;
706             } else if (expr_cols == 1) {
707                 row_slice_p = sel->slice1;
708                 col_slice_p = &zero_to_one;
709             } else {
710                 return new_val();
711             }
712             /* Alternately:

```



```

712     fprintf(stderr, "Runtime error: Only given one slice for a value with multiple
       rows and multiple columns\n");
713     debug_print(expr);
714     exit(-1); */
715 }
716 } else {
717     row_slice_p = sel->slice1;
718     col_slice_p = sel->slice2;
719 }
720 }
721 row_slice_success = resolve_rhs_slice(row_slice_p, expr_rows, r, &row_start, &
       row_end);
722 col_slice_success = resolve_rhs_slice(col_slice_p, expr_cols, c, &col_start, &
       col_end);
723 if (!row_slice_success || !col_slice_success) return new_val();
724 if (row_start < 0) row_start = 0;
725 if (col_start < 0) col_start = 0;
726 if (row_end > expr_rows) row_end = expr_rows;
727 if (col_end > expr_cols) col_end = expr_cols;
728 if (row_end <= row_start || col_end <= col_start) return new_val();
729 if (expr->flags == FLAG_NUMBER || expr->flags == FLAG_STRING) {
730     /* You would have thought we could figure this out a lot further up
731      * in the code, but had to be sure that (row_start, row_end, col_start, col_end)
732      * actually ended up as (0, 1, 0, 1) */
733     return clone_value(expr);
734 } else {
735     subrange.range = expr->subrange->range;
736     subrange.base_var_offset_row = expr->subrange->base_var_offset_row + row_start;
737     subrange.base_var_offset_col = expr->subrange->base_var_offset_col + col_start;
738     subrange.subrange_num_rows = row_end - row_start;
739     subrange.subrange_num_cols = col_end - col_start;
740     return deref_subrange_p(&subrange);
741 }
742 }
743
744 value_p getValSR(struct subrange_t *sr, int r, int c) {
745     if(sr->subrange_num_rows <= r || sr->subrange_num_cols <= c || r < 0 || c < 0)
746         return new_val();
747     return getVal(sr->range, r + sr->base_var_offset_row, c + sr->base_var_offset_col);
748 }
749
750 void verify_assert(value_p val, char *fname) {
751     if ((!assertSingleNumber(val)) || val->numericVal != 1.0) {
752         fprintf(stderr, "EXITING - The function %s was called with arguments of the wrong
       dimensions.\n", fname);
753         exit(-1);
754     }
755 }
756
757 value_p getVal(struct var_instance *inst, int r, int c) {
758     /* If we're going to return new_val() then we have to
759      * do clone_value(). Otherwise the receiver won't know
760      * whether or not they can free the value_p they get back.
761      * I think this should return, dangerously, return NULL if it's
762      * invalid, and the callers will have to be careful to check the value.
763      * The alternative is to always clone_value - safer, but much slower

```

```

764     * and makes our memory issues even bigger.
765     * Right now there are only a few places that call this. */
766
767 if(!assertInBounds(inst, r, c)) return NULL;
768 int cell_number = r * inst->cols + c;
769 char cell_status = inst->status[cell_number];
770 switch(cell_status) {
771     case NEVER_EXAMINED:
772         inst->status[cell_number] = IN_PROGRESS;
773         inst->values[cell_number] = calcVal(inst, r, c);
774         if (inst->values[cell_number]->flags == FLAG_SUBRANGE) {
775             int i, j;
776             for (i = 0; i < inst->values[cell_number]->subrange->subrange_num_rows; i++) {
777                 for (j = 0; j < inst->values[cell_number]->subrange->subrange_num_cols; j++)
778                     /* Prevent sneaky circular references */
779                     getVal(inst->values[cell_number]->subrange->range,
780                             i + inst->values[cell_number]->subrange->base_var_offset_row,
781                             j + inst->values[cell_number]->subrange->base_var_offset_col);
782             }
783         }
784     }
785     inst->status[cell_number] = CALCULATED;
786     break;
787 case IN_PROGRESS:
788     fprintf(stderr, "EXITING - Circular reference in %s[%d,%d]\n", inst->name, r, c)
789         ;
790     exit(-1);
791     break;
792 case CALCULATED:
793     if (inst->values[cell_number] == NULL) {
794         fprintf(stderr, "Supposedly, %s[%d,%d] was already calculated, but there is a
795             null pointer there.\n", inst->name, r, c);
796         fprintf(stderr, "Attempting to print contents of the variable instance where
797             this occurred:\n");
798         fflush(stderr);
799         debug_print_varinst(inst);
800         exit(-1);
801     }
802     break;
803 default:
804     fprintf(stderr, "Unrecognized cell status %d (row %d, col %d)!\n", cell_status,
805             r, c);
806     fprintf(stderr, "Attempting to print contents of the variable instance where
807         this occurred:\n");
808     fflush(stderr);
809     debug_print_varinst(inst);
810     exit(-1);
811     break;
812 }
813 return inst->values[cell_number];
814 }

```

## 9.12 stdlib.xtnd

```

1  /* jss2272 ns3158 */
2
3  global rounding_cutoff := 1e-7;
4  global digits_after_decimal := 6;
5
6  extern "stdlib.a" {
7      sin(val);
8      cos(val);
9      tan(val);
10     acos(val);
11     asin(val);
12     atan(val);
13     sinh(val);
14     cosh(val);
15     tanh(val);
16     exp(val);
17     log(val);
18     log10(val);
19     sqrt(val);
20     ceil(val);
21     fabs(val);
22     floor(val);
23     isNaN(val);
24     len(str);
25     round(val, number_of_digits);
26     isInfinite(val);
27     get_stdin();
28     get_stdout();
29     get_stderr();
30     open(filename, mode);
31     close(file_handle);
32     read(file_handle, num_bytes);
33     readline(file_handle);
34     write(file_handle, buffer);
35     toASCII(val);
36     fromASCII(val);
37     plot(val);
38     bar_chart(file_handle, labels, vals);
39     line_chart(file_handle, labels, x_vals);
40     parseFloat(val);
41 }
42
43 global STDIN := get_stdin();
44 global STDOUT := get_stdout();
45 global STDERR := get_stderr();
46
47 print_endline(val) {
48     return write(STDOUT, toString(val) + "\n");
49 }
50
51 transpose([m,n] rng) {
52     [n,m] ret := rng[column(),row()];
53     return ret;
54 }
55
56 flatten([m,n] rng) {

```

```

57     [1,m*n] ret := rng[floor(column()/n), column()%n];
58     return ret;
59 }
60
61 isNumber(x) {
62     return typeof(x) == "Number";
63 }
64
65 isEmpty(x) {
66     return typeof(x) == "Empty";
67 }
68
69 colRange(start, end) {
70     [end-start, 1] ret;
71     ret[0,0] = start;
72     ret[1:,0] = ret[[-1]] + 1;
73     return ret;
74 }
75
76 rowRange(start, end) {
77     return transpose(colRange(start,end));
78 }
79
80 matchCol([num_rows, 1] list, val) {
81     [num_rows, 1] amt_to_add, final_index;
82     amt_to_add[0,0] = val == #list ? 0 : 1;
83     amt_to_add[1:,0] = (amt_to_add[[-1]] == 0 || val == #list) ? 0 : 1;
84     final_index[0,0] = 0;
85     final_index[1:,0] = final_index[[-1]] + amt_to_add[[-1]];
86     return amt_to_add[-1] == 0 ? final_index[-1] : empty;
87 }
88
89 matchRow([1, num_cols] list, val) {
90     [1, num_cols] amt_to_add, final_index;
91     amt_to_add[0,0] = val == #list ? 0 : 1;
92     amt_to_add[0,1:] = (amt_to_add[[-1]] == 0 || val == #list) ? 0 : 1;
93     final_index[0,0] = 0;
94     final_index[0,1:] = final_index[[-1]] + amt_to_add[[-1]];
95     return amt_to_add[-1] == 0 ? final_index[-1] : empty;
96 }
97
98 match([m,n] list, val) {
99     return m == 1 ? matchRow(list, val) : (n == 1 ? matchCol(list, val) : empty);
100 }
101
102 bsearch([num_rows, 1] list, val) {
103     mid := (num_rows - 1) / 2;
104     return switch {
105         case list[mid] == val:
106             mid;
107         case list[mid] > val:
108             mid > 0 ? bsearch(list[:mid], val) : empty;
109         case list[mid] < val:
110             num_rows > 1 ? mid + 1 + bsearch(list[mid+1:], val) : empty;
111     };
112 }

```

```

113
114 sum_column([m,1] rng) {
115     [m,1] running_sum;
116     running_sum[0,0] = #rng;
117     running_sum[1:,0] = running_sum[[-1]] + #rng;
118     return running_sum[-1];
119 }
120
121 sum([m,n] rng) {
122     /* Returns the sum of the values in the range, skipping any values that are non-
        numeric */
123     [m,n] numbers := isNumber(#rng) ? #rng : 0;
124     [1,n] column_sums := sum_column(numbers[:,]);
125     return sum_column(transpose(column_sums));
126 }
127
128 nmax(n1, n2) {
129     return n1 > n2 ? n1 : n2;
130 }
131
132 max_column([m,1] rng) {
133     [m,1] running_max;
134     running_max[0,0] = #rng;
135     running_max[1:,0] = running_max[[-1]] > #rng ? running_max[[-1]] : #rng;
136     return running_max[-1];
137 }
138
139 max([m,n] rng) {
140     /* Returns the max of the values in the range, skipping any values that are non-
        numeric */
141     [m,n] numbers := isNumber(#rng) ? #rng : empty;
142     [1,n] column_maxs := max_column(rng[:,]);
143     return max_column(transpose(column_maxs));
144 }
145
146 nmin(n1, n2) {
147     return n1 < n2 ? n1 : n2;
148 }
149
150 min_column([m,1] rng) {
151     [m,1] running_min;
152     running_min[0,0] = #rng;
153     running_min[1:,0] = running_min[[-1]] > #rng ? running_min[[-1]] : #rng;
154     return running_min[-1];
155 }
156
157 min([m,n] rng) {
158     /* Returns the min of the values in the range, skipping any values that are non-
        numeric */
159     [m,n] numbers := isNumber(#rng) ? #rng : empty;
160     [1,n] column_mins := min_column(rng[:,]);
161     return min_column(transpose(column_mins));
162 }
163
164 sign(arg) {
165     return switch {

```

```

166     case arg > 0: 1;
167     case arg < 0: -1;
168     case arg == 0: 0;
169   };
170 }
171
172 gcd(m, n) {
173   return (n == 0) ? m : gcd(n, m % n);
174 }
175
176 lcm(m, n) {
177   return m * n / gcd(m, n);
178 }
179
180 sumsq([m,n] rng) {
181   [m,n] squares := #rng * #rng;
182   return sum(squares);
183 }
184
185 sumproduct([m,n] rng1, [m,n] rng2) {
186   [m,n] products := #rng1 * #rng2;
187   return sum(products);
188 }
189
190 sumxmy2([m,n] rng1, [m,n] rng2) {
191   [m,n] diffs := #rng1 - #rng2;
192   return sumsq(diffs);
193 }
194
195 mmult([m,n] rng1, [n,p] rng2) {
196   [m,p] result := sumproduct(rng1[:,:], transpose(rng2[:,:]));
197   return result;
198 }
199
200 linest([p,q] known_ys, [p,q] known_xs) {
201   flat_ys := flatten(known_ys);
202   flat_xs := flatten(known_xs);
203
204   n := p * q;
205   S_x := sum(flat_xs);
206   S_y := sum(flat_ys);
207   S_xx := sumsq(flat_xs);
208   S_yy := sumsq(flat_ys);
209   S_xy := sumproduct(flat_xs, flat_ys);
210
211   beta1_hat := (n * S_xy - S_x*S_y)/(n*S_xx - S_x*S_x);
212   beta0_hat := S_y / n - beta1_hat * S_x / n;
213   [2,2] ret;
214   ret[0,0] = "Intercept estimate";
215   ret[0,1] = "Slope estimate";
216   ret[1,0] = beta0_hat;
217   ret[1,1] = beta1_hat;
218   return ret;
219 }
220
221 toUpper(text) {

```

```

222   val := toASCII(text);
223   val_s := size(val);
224   [val_s[0],val_s[1]] result := #val >= 97 && #val <= 122 ? #val - 32 : #val;
225   return fromASCII(result);
226 }
227
228 toLower(text) {
229   val := toASCII(text);
230   val_s := size(val);
231   [val_s[0],val_s[1]] result := #val >= 65 && #val <= 90 ? #val + 32 : #val;
232   return fromASCII(result);
233 }
234
235 left(str, num_chars) {
236   return fromASCII(toASCII(str)[:num_chars]);
237 }
238
239 right(str, num_chars) {
240   return fromASCII(toASCII(str)[-num_chars:]);
241 }
242
243 substring(str, start, length) {
244   return fromASCII(toASCII(str)[start:start+length]);
245 }
246
247 concatRow([1,n] cells, joiner) {
248   [1,n] accum;
249   accum[0,0] = #cells;
250   accum[0,1:] = accum[-1] + joiner + #cells;
251   return accum[-1];
252 }
253
254 toRangeLiteral([m,n] rng) {
255   [m,n] strings := toLiteral(#rng);
256   [m,1] rows := concatRow(strings[:,], ", ");
257   return "{" + concatRow(transpose(rows), ";\\n") + "}";
258 }
259
260 toLiteral(arg) {
261   return switch(typeof(arg)) {
262     case "Number":
263       toString(arg);
264     case "String":
265       "\"" + arg + "\"";
266     case "Empty":
267       "empty";
268     case "Range":
269       toRangeLiteral(arg);
270   };
271 }
272
273 repeat(str, num) {
274   [1,num] copies := str;
275   return concatRow(copies,"");
276 }
277

```

```

278 stringOfPositiveInteger(arg) {
279     num_digits := 1 + floor(log10(arg));
280     [1,num_digits] digits := floor(arg/10**(num_digits-1-column())) % 10;
281     [1,num_digits] ascii_digits := 48 + #digits;
282     return arg < 1 ? "0" : fromASCII(ascii_digits);
283 }
284
285 padLeft(str, pad_char, total_length) {
286     existing_length := len(str);
287     padding := repeat(pad_char, total_length - len(str));
288     return existing_length < total_length ? (padding + str) : str;
289 }
290
291 toString(arg) {
292     positive_arg := fabs(arg);
293     closest_integer := round(positive_arg, 0);
294     is_integral_enough := fabs(positive_arg-closest_integer) < rounding_cutoff;
295     floating_part := round(10 ** digits_after_decimal * (positive_arg - floor(
        positive_arg)),0);
296     positive_part := stringOfPositiveInteger(floor(positive_arg)) + (is_integral_enough
        ? "" : "." + padLeft(stringOfPositiveInteger(floating_part), "0",
        digits_after_decimal));
297
298     return switch(typeof(arg)) {
299         case "Number":
300             switch {
301                 case isNaN(arg):
302                     "NaN";
303                 case isInfinite(arg) == -1:
304                     "-Inf";
305                 case isInfinite(arg) == 1:
306                     "Inf";
307                 case sign(arg) == 0:
308                     "0";
309                 case sign(arg) == 1:
310                     positive_part;
311                 case sign(arg) == -1:
312                     "-" + positive_part;
313                 default:
314                     "Encountered a number that is neither NaN, +Inf, -Inf, 0, positive or
                        negative";
315             };
316         case "String":
317             arg;
318         case "Empty":
319             "empty";
320         case "Range":
321             toRangeLiteral(arg);
322     };
323 }
324
325 numRows(arg) {
326     return size(arg)[0];
327 }
328
329 numCols(arg) {

```



```

330     return size(arg)[1];
331 }
332
333 splitChars([1,n] stringchars, splitchar) {
334     loc := matchRow(stringchars, splitchar);
335     firstword := fromASCII(stringchars[:loc]);
336     lastwords := splitChars(stringchars[loc+1:], splitchar);
337     combined := stack(firstword, lastwords);
338     return loc == empty ? fromASCII(stringchars) : combined;
339 }
340
341 split(string, splitter) {
342     return splitChars(toASCII(string), toASCII(splitter));
343 }
344
345 splitToRange(string, row_splitter, col_splitter) {
346     split_rows := split(string, row_splitter);
347     [numRows(split_rows),1] split_cols := split(#split_rows,col_splitter);
348     [numRows(split_rows),1] col_lengths := numRows(#split_cols);
349     [numRows(split_rows), max(col_lengths)] result := #split_cols[column()];
350     return result;
351 }
352
353 isSpace(char) {
354     return switch(char) {
355         case toASCII(" "), toASCII("\n"), toASCII("\t"), toASCII("\r"):
356             1;
357         default:
358             0;
359     };
360 }
361
362 trimChars(chars) {
363     return isSpace(chars[0]) ? trimChars(chars[1:]) : chars;
364 }
365
366 ltrim(s) {
367     return fromASCII(trimChars(toASCII(s)));
368 }
369
370 reverse(s) {
371     chars := toASCII(s);
372     l := len(s);
373     [1,numCols(chars)] chars_reversed := chars[l-1-column()];
374     return l ? fromASCII(chars_reversed) : "";
375 }
376
377 rtrim(s) {
378     return reverse(ltrim(reverse(s)));
379 }
380
381 trim(s) {
382     return ltrim(rtrim(s));
383 }
384
385 charAt(s, i) {

```

```

386     return toASCII(s)[i];
387 }
388
389 parseString(s) {
390     trimmed := trim(s);
391     rangeSplit := splitToRange(substring(trimmed, 1, len(trimmed) - 2), ";", ",");
392     [numRows(rangeSplit), numCols(rangeSplit)] rangeContents := parseString(#rangeSplit)
393     ;
394     return switch {
395         case charAt(trimmed,0) == toASCII("{") && charAt(trimmed,-1) == toASCII("}"):
396             rangeContents;
397         case charAt(trimmed,0) == toASCII("\") && charAt(trimmed,-1) == toASCII("\"):
398             substring(trimmed, 1, len(trimmed) - 2);
399         case trimmed == "empty":
400             empty;
401         default:
402             parseFloat(trimmed);
403     };
404 }
405
406 normalize([m,n] arg) {
407     [m,n] squared_lengths := #arg * #arg, normalized := #arg / vector_norm;
408     vector_norm := sqrt(sum(squared_lengths));
409     return normalized;
410 }
411
412 append([m,n] rg1, [p,q] rg2) {
413     [nmax(m,p), n+q] res;
414     res[:m,:n] = #rg1;
415     res[:p,n:n+q] = rg2[:,-n];
416     return res;
417 }
418
419 stack(rg1, rg2) {
420     return transpose(append(transpose(rg1), transpose(rg2)));
421 }

```

## 10. Tests and Output

helloworld.xtnd

```
1 main(args) {  
2   foo := print_endline("Hello World") -> 0;  
3   return foo;  
4 }
```

helloworld.xtnd - Expected Output

```
1 Hello World
```

test-access-cell.xtnd

```
1 main([1,n] args) {  
2   [2,2] foo := "string";  
3   bar := foo[1,1];  
4   return print_endline(bar) -> 0;  
5 }
```

test-access-cell.xtnd - Expected Output

```
1 string
```

test-access-column-cell.xtnd

```
1 main([1,n] args) {  
2   [4,1] foo := "string";  
3   return print_endline(foo[1,0]) -> 0;  
4 }
```

test-access-column-cell.xtnd - Expected Output

```
1 string
```

test-access-column-cells.xtnd

```
1 main([1,n] args) {  
2   [4,4] foo := "string";  
3   return print_endline( foo[2,:]) -> 0;  
4 }
```

test-access-column-cells.xtnd - Expected Output

```
1 {"string", "string", "string", "string"}
```

test-access-hashtag-multi-dim.xtnd

```

1 main([1,n] args) {
2   [4,4] foo := "string";
3   return print_endline( #foo) -> 0;
4 }

```

test-access-hashtag-multi-dim.xtnd - Expected Output

```

1 string

```

test-access-hashtag-single-dim.xtnd

```

1 main([1,n] args) {
2   [1,1] foo := "string";
3   return print_endline(#foo)-> 0;
4 }

```

test-access-hashtag-single-dim.xtnd - Expected Output

```

1 string

```

test-access-relative-range.xtnd

```

1 main([1,n] args) {
2   [4,4] foo := "string";
3   return print_endline( foo[, [1]]) -> 0;
4 }

```

test-access-relative-range.xtnd - Expected Output

```

1 string

```

test-access-selected-range-1.xtnd

```

1 main([1,n] args) {
2   [4,4] foo := "string";
3   return print_endline(foo[2: ,2:]) -> 0;
4 }

```

test-access-selected-range-1.xtnd - Expected Output

```

1 {"string", "string";
2 "string", "string"}

```

test-access-selected-range-2.xtnd

```

1 main([1,n] args) {
2   [4,4] foo := "string";
3   return print_endline(foo[2:3 ,2:4]) -> 0;
4 }

```

test-access-selected-range-2.xtnd - Expected Output

```

1 {"string", "string"}

```

test-access-x-range-of-cells.xtnd

```

1 main([1,n] args) {
2   [4,4] foo := "string";
3   return print_endline(foo[1,:]) -> 0;
4 }

```

#### test-access-x-range-of-cells.xtnd - Expected Output

```
1 {"string", "string", "string", "string"}
```

#### test-access-y-range-of-cells.xtnd

```
1 main([1,n] args) {  
2   [4,4] foo := "string";  
3   return print_endline( foo[:,1]) -> 0;  
4 }
```

#### test-access-y-range-of-cells.xtnd - Expected Output

```
1 {"string";  
2 "string";  
3 "string";  
4 "string"}
```

#### test-acos.xtnd

```
1 main(args) {  
2   return print_endline(acos(0.0)) -> 0;  
3 }
```

#### test-acos.xtnd - Expected Output

```
1 1.570796
```

#### test-addition.xtnd

```
1 main(args){  
2   return print_endline(5 + 7) -> 0;  
3 }
```

#### test-addition.xtnd - Expected Output

```
1 12
```

#### test-addition-empty.xtnd

```
1 main([1,1] args){  
2   return print_endline( empty + 5) -> 0;  
3 }
```

#### test-addition-empty.xtnd - Expected Output

```
1 empty
```

#### test-asin.xtnd

```
1 main([1,n] args) {  
2   return print_endline(asin(0.5)) -> 0;  
3 }
```

#### test-asin.xtnd - Expected Output

```
1 0.523599
```

#### test-atan.xtnd

```

1 main([1,n] args) {
2     return print_endline( atan(45.0)) -> 0;
3 }

```

**test-atan.xtnd - Expected Output**

```

1 1.548578

```

**test-basic-func.xtnd**

```

1 main([1,n] args) {
2     foo := 2;
3     bar := 3;
4     foobar := foo + bar;
5     return print_endline( 0) -> 0;
6 }

```

**test-basic-func.xtnd - Expected Output**

```

1 0

```

**test-bitnot.xtnd**

```

1 main(args) {
2     return print_endline(~{"a",1}) -> print_endline(~1) -> print_endline(~0) ->
        print_endline(~"a") -> print_endline(empty);
3 }

```

**test-bitnot.xtnd - Expected Output**

```

1 empty
2 -2
3 -1
4 empty
5 empty

```

**test-bitwise-and.xtnd**

```

1 main([1,1] args){
2     return print_endline(23 & 12) -> 0;
3 }

```

**test-bitwise-and.xtnd - Expected Output**

```

1 4

```

**test-bitwise-and-empty.xtnd**

```

1 main([1,1] args){
2     return print_endline(empty & 4) -> 0;
3 }

```

**test-bitwise-and-empty.xtnd - Expected Output**

```

1 empty

```

**test-bitwise-left.xtnd**

```

1 main([1,1] args){
2     return print_endline( 14 << 2) -> 0;
3 }

```

test-bitwise-left.xtnd - Expected Output

```

1 56

```

test-bitwise-left-empty.xtnd

```

1 main([1,1] args){
2     return print_endline( empty >> 1) -> 0;
3 }

```

test-bitwise-left-empty.xtnd - Expected Output

```

1 empty

```

test-bitwise-not.xtnd

```

1 main([1,1] args){
2     /* Should return -89 */
3     return print_endline(~88) -> 0;
4 }

```

test-bitwise-not.xtnd - Expected Output

```

1 -89

```

test-bitwise-not-empty.xtnd

```

1 main([1,1] args){
2     /* Should return empty */
3     return print_endline( ~empty) -> 0;
4 }

```

test-bitwise-not-empty.xtnd - Expected Output

```

1 empty

```

test-bitwise-or.xtnd

```

1 main([1,1] args){
2     return print_endline(14 | 12) -> 0;
3 }

```

test-bitwise-or.xtnd - Expected Output

```

1 14

```

test-bitwise-or-empty.xtnd

```

1 main([1,1] args){
2     return print_endline(empty | 2) -> 0;
3 }

```

test-bitwise-or-empty.xtnd - Expected Output

```

1 empty

```

test-bitwise-right.xtnd

```
1 main([1,1] args){
2   return print_endline(12 >> 2) -> 0;
3 }
```

test-bitwise-right.xtnd - Expected Output

```
1 3
```

test-bitwise-right-empty.xtnd

```
1 main([1,1] args){
2   return print_endline( empty >> 2) -> 0;
3 }
```

test-bitwise-right-empty.xtnd - Expected Output

```
1 empty
```

test-bitwise-xor.xtnd

```
1 main([1,1] args){
2   return print_endline(14 ^ 12) -> 0;
3 }
```

test-bitwise-xor.xtnd - Expected Output

```
1 2
```

test-bitwise-xor-empty.xtnd

```
1 main([1,1] args){
2   return print_endline(empty ^ 2) -> 0;
3 }
```

test-bitwise-xor-empty.xtnd - Expected Output

```
1 empty
```

test-boolean-equals.xtnd

```
1 main([1,1] args){
2   return print_endline( 5 == 6) -> 0;
3 }
```

test-boolean-equals.xtnd - Expected Output

```
1 0
```

test-boolean-equals-both-empty.xtnd

```
1 main([1,1] args){
2   return print_endline( empty == empty) -> 0;
3 }
```

test-boolean-equals-both-empty.xtnd - Expected Output

```
1 1
```



# test-boolean-equals-harder.xtnd

```
1 main([1,1] args){
2   return
3   print_endline( "True cases for ==") ->
4   print_endline(  (5 == 5)) ->
5   print_endline(  (5 == 5.0)) ->
6   print_endline(  (0.5 == 5e-1)) ->
7   print_endline(  (50 == 5e1)) ->
8   print_endline(  2 + 2 == 4) ->
9   print_endline(  "foo" == "foo") ->
10  print_endline(  "" == "") ->
11  print_endline(  empty == empty) ->
12  print_endline(  empty == !empty) ->
13  print_endline(  !"foo" == !"bar") ->
14  print_endline(  (2 ? 3 : 4) == ("foo" ? 3 : "not 4") ) ->
15
16  print_endline( "\nFalse cases for ==") ->
17  print_endline(  (5 == 6)) ->
18  print_endline(  (5 == 5.01)) ->
19  print_endline(  (0.5 == 5e-2)) ->
20  print_endline(  (50 == 5e2)) ->
21  print_endline(  2 + 2 == 5) ->
22  print_endline(  "foo" == "bar") ->
23  print_endline(  "" == "foo") ->
24  print_endline(  "" == empty) ->
25  print_endline(  2 == empty) ->
26  print_endline(  empty == 2) ->
27  print_endline(  (2 ? 3 : 4) == ("foo" ? "not 3" : 4) ) ->
28
29  print_endline( "\nTrue cases for !=") ->
30  print_endline(  (5 != 6)) ->
31  print_endline(  (5 != 5.01)) ->
32  print_endline(  (0.5 != 5e-2)) ->
33  print_endline(  (50 != 5e2)) ->
34  print_endline(  2 + 2 != 5) ->
35  print_endline(  "foo" != "bar") ->
36  print_endline(  "" != "foo") ->
37  print_endline(  "" != empty) ->
38  print_endline(  2 != empty) ->
39  print_endline(  empty != 2) ->
40  print_endline(  (2 ? 3 : 4) != ("foo" ? "not 3" : 4) ) ->
41
42  print_endline( "\nFalse cases for !=") ->
43  print_endline(  (5 != 5)) ->
44  print_endline(  (5 != 5.0)) ->
45  print_endline(  (0.5 != 5e-1)) ->
46  print_endline(  (50 != 5e1)) ->
47  print_endline(  2 + 2 != 4) ->
48  print_endline(  "foo" != "foo") ->
49  print_endline(  "" != "") ->
50  print_endline(  empty != empty) ->
51  print_endline(  empty != !empty) ->
52  print_endline(  !"foo" != !"bar") ->
53  print_endline(  (2 ? 3 : 4) != ("foo" ? 3 : "not 4") ) ->
54
55  0;
```

```
56 }
```

#### test-boolean-equals-harder.xtnd - Expected Output

```
1 True cases for ==
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13
14 False cases for ==
15 0
16 0
17 0
18 0
19 0
20 0
21 0
22 0
23 0
24 0
25 0
26
27 True cases for !=
28 1
29 1
30 1
31 1
32 1
33 1
34 1
35 1
36 1
37 1
38 1
39
40 False cases for !=
41 0
42 0
43 0
44 0
45 0
46 0
47 0
48 0
49 0
50 0
51 0
```

test-boolean-equals-one-empty.xtnd

```

1 main([1,1] args){
2     return print_endline( empty == 5) -> 0;
3 }

```

test-boolean-equals-one-empty.xtnd - Expected Output

```

1 0

```

test-boolean-logical-not-equals.xtnd

```

1 main([1,1] args){
2     return print_endline( 6 != 7) -> 0;
3 }

```

test-boolean-logical-not-equals.xtnd - Expected Output

```

1 1

```

test-boolean-logical-not-equals-both-empty.xtnd

```

1 main([1,1] args){
2     return print_endline( empty != empty) -> 0;
3 }

```

test-boolean-logical-not-equals-both-empty.xtnd - Expected Output

```

1 0

```

test-boolean-logical-not-equals-one-empty.xtnd

```

1 main([1,1] args){
2     return print_endline(empty != 5) -> 0;
3 }

```

test-boolean-logical-not-equals-one-empty.xtnd - Expected Output

```

1 1

```

test-calling-func-from-import.xtnd

```

1 main([1,n] args){
2     return print_endline( gcd(70, 55)) -> 0;
3 }

```

test-calling-func-from-import.xtnd - Expected Output

```

1 5

```

test-ceil.xtnd

```

1 main([1,n] args) {
2     return print_endline(ceil(10.45)) -> 0;
3 }

```

test-ceil.xtnd - Expected Output

```

1 11

```

test-cos.xtnd

```
1 main([1,n] args) {
2     return print_endline(cos(45.0)) -> 0;
3 }
```

**test-cos.xtnd - Expected Output**

```
1 0.525322
```

**test-cosh.xtnd**

```
1 main([1,n] args) {
2     return print_endline( cosh(2.5)) -> 0;
3 }
```

**test-cosh.xtnd - Expected Output**

```
1 6.132289
```

**test-division.xtnd**

```
1 main([1,1] args){
2     /* Should evaluate to 4 */
3     return print_endline( 20 / 5) -> 0;
4 }
```

**test-division.xtnd - Expected Output**

```
1 4
```

**test-division-empty.xtnd**

```
1 main([1,n] args){
2     /* Should return empty */
3     return print_endline( empty / 5) -> 0;
4 }
```

**test-division-empty.xtnd - Expected Output**

```
1 empty
```

**test-exp.xtnd**

```
1 main([1,n] args) {
2     return print_endline(exp(2.0)) -> 0;
3 }
```

**test-exp.xtnd - Expected Output**

```
1 7.389056
```

**test-fabs.xtnd**

```
1 main([1,n] args) {
2     return print_endline(fabs(-45.0)) -> 0;
3 }
```

**test-fabs.xtnd - Expected Output**

```
1 45
```

#### test-file-close.xtnd

```
1 main(args){
2     return close(open("testcases/assets/test_file.txt", "r")) -> print_endline("Made it
   this far") -> 0;
3 }
```

#### test-file-close.xtnd - Expected Output

```
1 Made it this far
```

#### test-file-read.xtnd

```
1 main(args){
2     return print_endline(read(open("testcases/assets/test_file.txt", "r"),5)) -> 0;
3 }
```

#### test-file-read.xtnd - Expected Output

```
1 This
```

#### test-file-slurp.xtnd

```
1 main(args){
2     return
3     write(STDOUT, (read(open("testcases/assets/test_file.txt", "r"),0))) ->
4     0;
5 }
```

#### test-file-slurp.xtnd - Expected Output

```
1 This is a test file!
```

#### test-file-write.xtnd

```
1 main(args){
2     handle := open("testcases/assets/test_file_write.out", "w");
3     return
4     write(handle, "Hello") ->
5     close(handle) ->
6     print_endline("Made it this far") ->
7     0;
8 }
```

#### test-file-write.xtnd - Expected Output

```
1 Made it this far
```

#### test-floor.xtnd

```
1 main([1,n] args) {
2     return print_endline(floor(10.45)) -> 0;
3 }
```

#### test-floor.xtnd - Expected Output

```
1 10
```

#### test-func-params.xtnd

```

1 main([1,n] args) {
2     return print_endline( foo("string")) -> 0;
3 }
4 foo([1,1] arg) {
5     return arg;
6 }

```

**test-func-params.xtnd - Expected Output**

```

1 string

```

**test-func-params-omit-dim.xtnd**

```

1 main([1,n] args) {
2     return print_endline(foo("string")) -> 0;
3 }
4 foo([1,1] arg) {
5     return arg;
6 }

```

**test-func-params-omit-dim.xtnd - Expected Output**

```

1 string

```

**test-global-hello.xtnd**

```

1 bar() {
2     foo := 5;
3     return 2;
4 }
5
6 global foo := print_endline("Hello Globals!") -> 0;
7
8 main(args) {
9     return foo;
10 }

```

**test-global-hello.xtnd - Expected Output**

```

1 Hello Globals!

```

**test-global-masking.xtnd**

```

1 bar() {
2     foo := 5;
3     return 2;
4 }
5
6 global foo := print_endline("Hello Globals!") -> 0;
7
8 main(args) {
9     foo := print_endline("Hello Locals!") -> 0;
10    return foo;
11 }

```

**test-global-masking.xtnd - Expected Output**

```

1 Hello Locals!

```

#### test-globals.xtnd

```
1 global [2,2] foo := 1;
2 main([1,n] args) {
3     return print_endline(foo) -> 0;
4 }
```

#### test-globals.xtnd - Expected Output

```
1 {1, 1;
2 1, 1}
```

#### test-globals-between-imports.xtnd

```
1 import "../testcases/assets/string.xtnd";
2 global foo;
3 global [2, 5] bar;
4 import "../testcases/assets/string.xtnd";
```

#### test-globals-between-imports.xtnd - Expected Output

```
1 Hello
```

#### test-greater-than.xtnd

```
1 main([1,1] args){
2     return print_endline( 6 > 5) -> 0;
3 }
```

#### test-greater-than.xtnd - Expected Output

```
1 1
```

#### test-greater-than-empty.xtnd

```
1 main([1,1] args){
2     return print_endline( empty > 5) -> 0;
3 }
```

#### test-greater-than-empty.xtnd - Expected Output

```
1 empty
```

#### test-greater-than-or-equal.xtnd

```
1 main([1,1] args){
2     return print_endline( 7 >= 7) -> 0;
3 }
```

#### test-greater-than-or-equal.xtnd - Expected Output

```
1 1
```

#### test-greater-than-or-equal-empty.xtnd

```
1 main([1,1] args){
2     return print_endline(empty >= 7) -> 0;
3 }
```

#### test-greater-than-or-equal-empty.xtnd - Expected Output

```
1 empty
```

#### **test-less-than.xtnd**

```
1 main([1,1] args){
2   return print_endline( 6 < 7) -> 0;
3 }
```

#### **test-less-than.xtnd - Expected Output**

```
1 1
```

#### **test-less-than-empty.xtnd**

```
1 main([1,1] args){
2   return print_endline( empty > 5) -> 0;
3 }
```

#### **test-less-than-empty.xtnd - Expected Output**

```
1 empty
```

#### **test-less-than-or-equal.xtnd**

```
1 main([1,1] args){
2   return print_endline( 7 <= 5) -> 0;
3 }
```

#### **test-less-than-or-equal.xtnd - Expected Output**

```
1 0
```

#### **test-less-than-or-equal-empty.xtnd**

```
1 main([1,1] args){
2   return print_endline(empty <= 8) -> 0;
3 }
```

#### **test-less-than-or-equal-empty.xtnd - Expected Output**

```
1 empty
```

#### **test-log.xtnd**

```
1 main([1,n] args) {
2   return print_endline(log(10.0)) -> 0;
3 }
```

#### **test-log.xtnd - Expected Output**

```
1 2.302585
```

#### **test-log10.xtnd**

```
1 main([1,n] args) {
2   return print_endline( log10(100.0)) -> 0;
3 }
```

#### **test-log10.xtnd - Expected Output**



```
1 2
```

#### `test-logical-and.xtnd`

```
1 main([1,1] args){
2     return print_endline( 1 && 6) -> 0;
3 }
```

#### `test-logical-and.xtnd - Expected Output`

```
1 1
```

#### `test-logical-and-empty.xtnd`

```
1 main([1,1] args){
2     return print_endline( empty && 1) -> 0;
3 }
```

#### `test-logical-and-empty.xtnd - Expected Output`

```
1 empty
```

#### `test-logical-not.xtnd`

```
1 main([1,1] args){
2     return print_endline( !5) -> 0;
3 }
```

#### `test-logical-not.xtnd - Expected Output`

```
1 0
```

#### `test-logical-not-empty.xtnd`

```
1 main([1,1] args){
2     return print_endline( !empty) -> 0;
3 }
```

#### `test-logical-not-empty.xtnd - Expected Output`

```
1 empty
```

#### `test-logical-or.xtnd`

```
1 main([1,1] args){
2     return print_endline( 5 || 6) -> 0;
3 }
```

#### `test-logical-or.xtnd - Expected Output`

```
1 1
```

#### `test-logical-or-empty.xtnd`

```
1 main([1,1] args){
2     return print_endline( empty || 4) -> 0;
3 }
```

#### `test-logical-or-empty.xtnd - Expected Output`

```
1 empty
```

#### **test-modulo.xtnd**

```
1 main([1,n] args){
2     /* Should return 1 */
3     return print_endline(5 % 4) -> 0;
4 }
```

#### **test-modulo.xtnd - Expected Output**

```
1 1
```

#### **test-modulo-empty.xtnd**

```
1 main([1,n] args){
2     /* Should return empty */
3     return print_endline( empty % 5) -> 0;
4 }
```

#### **test-modulo-empty.xtnd - Expected Output**

```
1 empty
```

#### **test-multiple-imports.xtnd**

```
1 import "../testcases/assets/string.xtnd";
2 import "../testcases/assets/string.xtnd";
```

#### **test-multiple-imports.xtnd - Expected Output**

```
1 Hello
```

#### **test-multiplication.xtnd**

```
1 main([1,n] args){
2     /* Should evaluate to 35 */
3     return print_endline(7 * 5) -> 0;
4 }
```

#### **test-multiplication.xtnd - Expected Output**

```
1 35
```

#### **test-multiplication-empty.xtnd**

```
1 main([1,n] args){
2     /* Should evaluate to empty */
3     return print_endline(empty * 5) -> 0;
4 }
```

#### **test-multiplication-empty.xtnd - Expected Output**

```
1 empty
```

#### **test-nan-and-infinity.xtnd**

```

1  main(args) {
2      should_be_nan := sqrt(-1);
3      should_also_be_nan := 0 / 0;
4      should_be_plus_inf := 2 / 0;
5      should_be_minus_inf := -3 / 0;
6      should_be_normal := 4;
7      foo := "Hello";
8      bar := empty;
9      [3,3] baz := row() * column();
10
11     return
12         print_endline(typeof(should_be_nan)) -> // "Number"
13         print_endline(typeof(should_also_be_nan)) -> // "Number"
14         print_endline(typeof(should_be_plus_inf)) -> // "Number"
15         print_endline(typeof(should_be_minus_inf)) -> // "Number"
16         print_endline(typeof(should_be_normal)) -> // "Number"
17         print_endline(typeof(foo)) -> // "String"
18         print_endline(typeof(bar)) -> // "Empty"
19         print_endline(typeof(baz)) -> // "Range"
20         print_endline("") ->
21
22         print_endline(isNaN(should_be_nan)) -> // 1
23         print_endline(isNaN(should_also_be_nan)) -> // 1
24         print_endline(isNaN(should_be_plus_inf)) -> // 0
25         print_endline(isNaN(should_be_minus_inf)) -> // 0
26         print_endline(isNaN(should_be_normal)) -> // 0
27         print_endline(isNaN(foo)) -> // 0
28         print_endline(isNaN(bar)) -> // 0
29         print_endline(isNaN(baz)) -> // 0
30         print_endline("") ->
31
32         print_endline(isInfinite(should_be_nan)) -> // 0
33         print_endline(isInfinite(should_also_be_nan)) -> // 0
34         print_endline(isInfinite(should_be_plus_inf)) -> // 1
35         print_endline(isInfinite(should_be_minus_inf)) -> // -1
36         print_endline(isInfinite(should_be_normal)) -> // 0
37         print_endline(isInfinite(foo)) -> // 0
38         print_endline(isInfinite(bar)) -> // 0
39         print_endline(isInfinite(baz)) -> // 0
40
41     0;
42 }

```

#### test-nan-and-infinity.xtnd - Expected Output

```

1  Number
2  Number
3  Number
4  Number
5  Number
6  String
7  Empty
8  Range
9
10 1
11 1
12 0

```

```
13 0
14 0
15 empty
16 empty
17 empty
18
19 0
20 0
21 1
22 -1
23 0
24 empty
25 empty
26 empty
```

#### **test-parse-error.xtnd**

```
1 main(args){
2     foo := 5$5;
3     return foo;
4 }
```

#### **test-parse-error.xtnd - Expected Output**

```
1 Syntax error in "./testcases/inputs_regression/test_parse_error.xtnd": Invalid
   character: $
2 Line 2 at character 11
```

#### **test-parse-error-after-multiline-comment.xtnd**

```
1 main(args){
2     /* This is a comment spanning multiple lines.
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 20 of them, in fact. */
22     foo := 5/5;
23     bar := $$$$
24     return foo;
25 }
```

#### **test-parse-error-after-multiline-comment.xtnd - Expected Output**

```
1 Syntax error in "./testcases/inputs_regression/
    test_parse_error_after_multiline_comment.xtnd": Invalid character: $
2 Line 23 at character 10
```

```
1 main(args){
2     foo := 5/5;
3     /* Test comment */ foo := 5$5;
4     return foo;
5 }
```

```
1 Syntax error in "./testcases/inputs_regression/test_parse_error_comment.xtnd": Invalid
  character: $
2 Line 3 at character 30
```

```
1 main([1,1] args){
2   x := switch() {
3     case 1 > 2: 100;
4     case 3 > 0: 200
5   };
6   return print_endline(x) -> 0;
7 }
```

```
1 Syntax error in "./testcases/inputs_regression/test_parse_error_missing_semicolon.xtnd
   ":
2 Line 5 at character 2
```

[illegible]

```
1 Syntax error in "./testcases/inputs_regression/test_parse_error_newlines.xtnd":
    Invalid character: $
2 Line 3 at character 52
```

```
1  main(args){
2      foo := "Hello";$$$;
3      return foo;
4  }
```

#### test-power.xtnd

```
1 main([1,n] args){
2     /* Should return 216 */
3     return print_endline(6**3) -> 0;
4 }
```

#### test-power.xtnd - Expected Output

```
1 216
```

#### test-power-empty.xtnd

```
1 main([1,n] args){
2     /* Should return empty */
3     return print_endline(empty**5) -> 0;
4 }
```

#### test-power-empty.xtnd - Expected Output

```
1 empty
```

#### test-print-empty.xtnd

```
1 main([1,n] args) {
2     foo := empty;
3     return print_endline( foo) -> 0;
4 }
```

#### test-print-empty.xtnd - Expected Output

```
1 empty
```

#### test-print-multi-range.xtnd

```
1 main([1,n] args) {
2     [5,5] foo := 1;
3     return print_endline( foo) -> 0;
4 }
```

#### test-print-multi-range.xtnd - Expected Output

```
1 {1, 1, 1, 1, 1;
2 1, 1, 1, 1, 1;
3 1, 1, 1, 1, 1;
4 1, 1, 1, 1, 1;
5 1, 1, 1, 1, 1}
```

#### test-print-multi-str-range.xtnd

```
1 main([1,n] args) {
2     [1,5] foo := "string";
3     return print_endline( foo) -> 0;
4 }
```

#### test-print-multi-str-range.xtnd - Expected Output

```
1 {"string", "string", "string", "string", "string"}
```

#### test-print-nums.xtnd

```

1 main([1,n] args) {
2   foo := 1;
3   return print_endline(foo) -> 0;
4 }

```

test-print-nums.xtnd - Expected Output

```

1 1

```

test-print-oned-range.xtnd

```

1 main([1,n] args) {
2   [1,10] foo := 1;
3   return print_endline( foo) -> 0;
4 }

```

test-print-oned-range.xtnd - Expected Output

```

1 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

```

test-print-str.xtnd

```

1 main([1,n] args) {
2   foo := "string";
3   return print_endline(foo) -> 0;
4 }

```

test-print-str.xtnd - Expected Output

```

1 string

```

test-range-equality.xtnd

```

1 main(args) {
2   my1 := {"Hello, world", "Goodbye, world"};
3   my2 := {"Hello, world", "Goodbye, world"};
4   my3 := {3,4,5,{"Hello, world", "Goodbye, world"},6,7,8};
5   my4 := {3,empty,5,{"Hello, world", "Goodbye, world"},6,7,8};
6   my5 := {3,4,5,{"Hello, world", "Goodbye, world"},6,7,8};
7   [2,2] foo := my1;
8   [2,1] bar := my1;
9   [3,3] ident := row() == column();
10  ident_lit := {1,0,0;0,1,0;0,0,1};
11  [3,3] all_ones := 1;
12  baz := my2;
13  return
14    // True cases
15    print_endline(my1 == my2) ->
16    print_endline(baz == my1) ->
17    print_endline(foo[0,0] == my2) ->
18    print_endline(foo[0,1] == my2) ->
19    print_endline(foo[0,0] == foo[1,1]) ->
20    print_endline(foo[:,0] == bar) ->
21    print_endline(my3[3] == my1) ->
22    print_endline(ident == ident_lit) ->
23    print_endline("") ->
24
25    // False cases

```

```

26     print_endline(my3 == my5) ->
27     print_endline(my3 == my4) ->
28     print_endline(foo == bar) ->
29     print_endline(foo == foo[0,0]) ->
30     print_endline(ident == all_ones) ->
31     print_endline(ident == 1) ->
32     print_endline(all_ones == 1) ->
33     0
34     ;
35 }

```

#### test-range-equality.xtnd - Expected Output

```

1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9
10 0
11 0
12 0
13 0
14 0
15 0
16 0

```

#### test-ref-between-globals.xtnd

```

1 global [2,2] foo;
2 global [2,2] bar;
3 main([1,n] args) {
4     foo := 1;
5     bar := foo;
6     return print_endline(bar) -> 0;
7 }

```

#### test-ref-between-globals.xtnd - Expected Output

```

1 1

```

#### test-short-circuiting-and.xtnd

```

1 main([1,1] args){
2     return 0 && print_endline("FAIL") -> print_endline("PASS") -> 0;
3 }

```

#### test-short-circuiting-and.xtnd - Expected Output

```

1 PASS

```

#### test-short-circuiting-and2.xtnd

```

1 main([1,1] args){
2     return 1 && print_endline("PASS1") -> print_endline("PASS2") -> 0;
3 }

```



#### test-short-circuiting-and2.xtnd - Expected Output

```
1 PASS1
2 PASS2
```

#### test-short-circuiting-or.xtnd

```
1 main([1,1] args){
2   return 0 || print_endline("PASS1") -> print_endline("PASS2") -> 0;
3 }
```

#### test-short-circuiting-or.xtnd - Expected Output

```
1 PASS1
2 PASS2
```

#### test-short-circuiting-or2.xtnd

```
1 main([1,1] args){
2   return 1 || print_endline("FAIL") -> print_endline("PASS") -> 0;
3 }
```

#### test-short-circuiting-or2.xtnd - Expected Output

```
1 PASS
```

#### test-signature-vars.xtnd

```
1 foo([m,n] arg) {
2   return "I was called with an argument with " + toString(m) + " rows and " + toString
      (n) + " columns.";
3 }
4
5 main([1,1] args) {
6   [42,17] x;
7   return print_endline(foo(x)) -> 0;
8 }
```

#### test-signature-vars.xtnd - Expected Output

```
1 I was called with an argument with 42 rows and 17 columns.
```

#### test-sin.xtnd

```
1 main([1,n] args) {
2   return print_endline(sin(45.0)) -> 0;
3 }
```

#### test-sin.xtnd - Expected Output

```
1 0.850904
```

#### test-sin-through-function.xtnd

```
1 internal_sin(x,y,z) {
2   return sin(z);
3 }
4
5 main([1,n] args) {
6   return print_endline(internal_sin(1,2,45.0)) -> 0;
7 }
```

#### test-sin-through-function.xtnd - Expected Output

```
1 0.850904
```

#### test-sin-through-function-and-global.xtnd

```
1 global theta := 45.0;
2
3 internal_sin(x,y,z) {
4     return sin(z);
5 }
6
7 main([1,n] args) {
8     return print_endline(internal_sin(1,2,theta)) -> 0;
9 }
```

#### test-sin-through-function-and-global.xtnd - Expected Output

```
1 0.850904
```

#### test-single-import.xtnd

```
1 main([1,n] args) {
2     return print_endline(gcd(70, 55)) -> 0;
3 }
```

#### test-single-import.xtnd - Expected Output

```
1 5
```

#### test-sinh.xtnd

```
1 main([1,n] args) {
2     return print_endline(sinh(3.0)) -> 0;
3 }
```

#### test-sinh.xtnd - Expected Output

```
1 10.017875
```

#### test-sqrt.xtnd

```
1 main([1,n] args) {
2     return print_endline(sqrt(9.0)) -> 0;
3 }
```

#### test-sqrt.xtnd - Expected Output

```
1 3
```

#### test-string-concatenation.xtnd

```
1 main(args) {
2     foo :=
3     print_endline("Hello " + "World") ->
4     print_endline("Hello " + "World") ->
5     print_endline("Hello " + ("World" + "")) ->
6     0;
7     return foo;
8 }
```

#### test-string-concatenation.xtnd - Expected Output

```
1 Hello World
2 Hello World
3 Hello World
```

#### test-subtraction.xtnd

```
1 main([1,1] args){
2   return print_endline( 7 - 5) -> 0;
3 }
```

#### test-subtraction.xtnd - Expected Output

```
1 2
```

#### test-subtraction-empty.xtnd

```
1 main([1,1] args){
2   return print_endline( empty - 2) -> 0;
3 }
```

#### test-subtraction-empty.xtnd - Expected Output

```
1 empty
```

#### test-switch-v1.xtnd

```
1 main([1,1] args){
2   x := switch(1) {
3     case 1: 100;
4     case 2: 200;
5     default: 300;
6   };
7   return print_endline(x) -> 0;
8 }
```

#### test-switch-v1.xtnd - Expected Output

```
1 100
```

#### test-switch-v10.xtnd

```
1 main([1,1] args){
2   x := switch {
3     case 0: 100;
4     case "also true": 200;
5     default: 99;
6   };
7   return print_endline(x) -> 0;
8 }
```

#### test-switch-v10.xtnd - Expected Output

```
1 200
```

#### test-switch-v11.xtnd

```

1 main([1,1] args){
2   x := switch {
3     case 0: 100;
4     default: 99;
5   };
6   return print_endline(x) -> 0;
7 }

```

test-switch-v11.xtnd - Expected Output

```

1 99

```

test-switch-v2.xtnd

```

1 main([1,1] args){
2   x := switch(2) {
3     case 1: 100;
4     case 2: 200;
5     default: 300;
6   };
7   return print_endline(x) -> 0;
8 }

```

test-switch-v2.xtnd - Expected Output

```

1 200

```

test-switch-v3.xtnd

```

1 main([1,1] args){
2   x := switch(3) {
3     case 1: 100;
4     case 2: 200;
5     default: 300;
6   };
7   return print_endline(x) -> 0;
8 }

```

test-switch-v3.xtnd - Expected Output

```

1 300

```

test-switch-v4.xtnd

```

1 main([1,1] args){
2   x := switch(2) {
3     case 1, 2: 100;
4     default: 300;
5   };
6   return print_endline(x) -> 0;
7 }

```

test-switch-v4.xtnd - Expected Output

```

1 100

```

test-switch-v5.xtnd

```

1 main([1,1] args){
2   x := switch(3) {
3     case 1, 2: 100;
4     default: 300;
5   };
6   return print_endline(x) -> 0;
7 }

```

test-switch-v5.xtnd - Expected Output

```

1 300

```

test-switch-v6.xtnd

```

1 main([1,1] args){
2   x := switch(3) {
3     case 1, 2: 100;
4     case 0, 3: 200;
5     default: 300;
6   };
7   return print_endline(x) -> 0;
8 }

```

test-switch-v6.xtnd - Expected Output

```

1 200

```

test-switch-v7.xtnd

```

1 main([1,1] args){
2   x := switch(4) {
3     case 1, 2: 100;
4     case 0, 3: 200;
5   };
6   return print_endline(x) -> 0;
7 }

```

test-switch-v7.xtnd - Expected Output

```

1 empty

```

test-switch-v8.xtnd

```

1 main([1,1] args){
2   x := switch() {
3     case 1 > 2: 100;
4     case 3 > 0: 200;
5   };
6   return print_endline(x) -> 0;
7 }

```

test-switch-v8.xtnd - Expected Output

```

1 200

```

test-switch-v9.xtnd

```

1 main([1,1] args){
2     x := switch {
3         case "true": 100;
4         case "also true": 200;
5     };
6     return print_endline(x) -> 0;
7 }

```

**test-switch-v9.xtnd - Expected Output**

```

1 100

```

**test-tan.xtnd**

```

1 main([1,n] args) {
2     return print_endline(tan(45.0)) -> 0;
3 }

```

**test-tan.xtnd - Expected Output**

```

1 1.619775

```

**test-tanh.xtnd**

```

1 main([1,n] args) {
2     return print_endline(tanh(45.0)) -> 0;
3 }

```

**test-tanh.xtnd - Expected Output**

```

1 1

```

**test-ternary-conditional.xtnd**

```

1 main([1,1] args){
2     return print_endline(5 ? 2 : 3) -> 0;
3 }

```

**test-ternary-conditional.xtnd - Expected Output**

```

1 2

```

**test-ternary-conditional-empty.xtnd**

```

1 main([1,1] args){
2     return print_endline( empty ? 5 : 6) -> 0;
3 }

```

**test-ternary-conditional-empty.xtnd - Expected Output**

```

1 empty

```

**test-unary-negation.xtnd**

```

1 main([1,n] args){
2     /* Should return -33 */
3     return print_endline( -33) -> 0;
4 }

```

**test-unary-negation.xtnd - Expected Output**

```
1 -33
```

```
test-unary-negation-empty.xtnd
```

```
1 main([1,n] args){  
2     return print_endline(-empty) -> 0;  
3 }
```

```
test-unary-negation-empty.xtnd - Expected Output
```

```
1 empty
```

## 11. Git Logs

```
1  a6db867 2016-12-19T20:36:52-05:00 GitHub: Merge pull request #135 from ExtendLang/
   contrib
2  900411c 2016-12-19T17:40:28-05:00 GitHub: Merge branch 'master' into contrib
3  d253e33 2016-12-19T17:39:19-05:00 Nigel Schuster: Assigned contributions
4  2569ea5 2016-12-19T12:43:20-05:00 GitHub: Merge pull request #134 from ExtendLang/
   stdlib-additions
5  47f1167 2016-12-19T12:09:58-05:00 GitHub: Merge branch 'master' into stdlib-additions
6  2e18028 2016-12-19T12:09:50-05:00 GitHub: Merge pull request #133 from ExtendLang/
   mergesort
7  142938e 2016-12-19T11:51:25-05:00 oracleofnj: Couple stdlib additions
8  3fc2843 2016-12-19T10:17:09-05:00 Nigel Schuster: Mergesort example
9  0b23496 2016-12-19T09:07:15-05:00 GitHub: Merge pull request #132 from ExtendLang/
   fixing-tcs
10 b86d992 2016-12-19T00:38:27-05:00 oracleofnj: Verify samples, compact TSP
11 e859c75 2016-12-19T00:17:57-05:00 oracleofnj: Final test run with complete stdlib
12 f02f2da 2016-12-19T00:13:20-05:00 oracleofnj: Remove obsolete print flavors
13 dd65154 2016-12-19T00:08:17-05:00 oracleofnj: Merge branch 'master' into fixing-tcs
14 82f4ad5 2016-12-19T00:07:45-05:00 oracleofnj: Merge branch 'master' of https://github.
   com/ExtendLang/Extend
15 d2cd19e 2016-12-19T00:07:30-05:00 oracleofnj: 125 / 125
16 8d02537 2016-12-19T00:07:11-05:00 GitHub: Merge pull request #131 from ExtendLang/
   webgif
17 453b7f6 2016-12-18T23:33:06-05:00 oracleofnj: Back to passing all previously passing
   TCs; on to stragglers
18 6487840 2016-12-18T23:00:01-05:00 Nigel Schuster: Removed webgif from Makefile
19 84dca34 2016-12-18T22:29:59-05:00 oracleofnj: Ignore webgif
20 ed204c2 2016-12-18T22:29:32-05:00 oracleofnj: Ignore webgif
21 1bcb830 2016-12-18T19:48:02-05:00 GitHub: Merge pull request #41 from ExtendLang/
   plotting
22 1cd2360 2016-12-18T19:46:21-05:00 GitHub: Merge branch 'master' into plotting
23 0058659 2016-12-18T19:30:35-05:00 GitHub: Merge pull request #129 from ExtendLang/
   remove-debug-final
24 b54f4aa 2016-12-18T18:54:47-05:00 GitHub: Fix MAXFLOAT
25 9324fb8 2016-12-18T18:50:59-05:00 GitHub: Merge branch 'master' into plotting
26 f152bc9 2016-12-18T18:32:21-05:00 oracleofnj: Remove Debug()
27 c069630 2016-12-18T18:15:40-05:00 Nigel Schuster: Linking plotter is optional
28 e9dbd0f 2016-12-18T17:05:54-05:00 GitHub: Merge pull request #128 from ExtendLang/back
   -to-parsing
29 b602263 2016-12-18T14:12:48-05:00 oracleofnj: Merge in cool program
30 45d14cf 2016-12-18T14:12:28-05:00 GitHub: Merge pull request #127 from ExtendLang/
   strcat-bug
31 790bc51 2016-12-18T14:00:48-05:00 oracleofnj: Merge branch 'cool_program' into back-to
   -parsing
32 2eba5e5 2016-12-18T13:59:44-05:00 oracleofnj: Replace C extend_parseString with in-
```



```

    language parseString
33 1b664be 2016-12-18T09:14:51-05:00 Nigel Schuster: Corrected travis file
34 554b584 2016-12-18T09:13:54-05:00 Nigel Schuster: Cleand up Makefile mess
35 a222916 2016-12-18T08:19:54-05:00 GitHub: Merge branch 'master' into plotting
36 d064d8a 2016-12-18T01:49:39-05:00 Ishaan: Cleanup line function
37 47dace5 2016-12-18T01:48:28-05:00 Ishaan: Test single parameter line chart
38 8f5cf52 2016-12-18T01:43:49-05:00 Ishaan: Fix the testcase fail
39 84cd775 2016-12-18T01:41:50-05:00 Ishaan: update testcase
40 a425775 2016-12-18T01:39:43-05:00 Ishaan: Figure out 2 line issue
41 ba2c3c1 2016-12-18T01:34:34-05:00 Ishaan: Add y values and update testcase
42 7ad5986 2016-12-18T01:18:28-05:00 Ishaan: Trying another version of line
43 b8732dd 2016-12-18T00:42:25-05:00 Ishaan: Fix derp in linechart
44 20e2c43 2016-12-18T00:40:22-05:00 Ishaan: Added basic linechart function to examine
45 b404e12 2016-12-17T23:56:16-05:00 Ishaan: Cast to float
46 e866f68 2016-12-17T23:38:40-05:00 Ishaan: Reverse row and col
47 8419510 2016-12-17T23:27:13-05:00 oracleofnj: That's a wrap
48 6ec3e0e 2016-12-17T23:17:46-05:00 oracleofnj: Proof of concept
49 302af00 2016-12-17T23:09:14-05:00 Ishaan: Updating checks
50 7b09def 2016-12-17T23:03:14-05:00 Ishaan: Testing bar chart plotting, will clean up
    later
51 20adaca 2016-12-17T20:40:13-05:00 oracleofnj: Some bugfixes
52 ad69dcf 2016-12-17T20:12:33-05:00 oracleofnj: Fixed extend side
53 8f76e59 2016-12-17T20:10:06-05:00 Kevin: Fixed highest_tsp to take in any number of
    players
54 0707084 2016-12-17T19:53:58-05:00 oracleofnj: Isolating
55 4479213 2016-12-17T19:47:38-05:00 oracleofnj: much longer
56 4be857f 2016-12-17T19:41:59-05:00 oracleofnj: seg fault
57 74358c1 2016-12-17T19:34:18-05:00 Kevin: Interesting program in Extend
58 ab1e1d2 2016-12-17T16:12:00-05:00 oracleofnj: Add some more stdlib funcs
59 b53463d 2016-12-17T14:51:24-05:00 GitHub: Merge pull request #125 from ExtendLang/
    stdlib-string
60 39046bc 2016-12-17T14:39:54-05:00 oracleofnj: Merge branch 'master' into stdlib-string
61 a01cc84 2016-12-17T14:39:40-05:00 oracleofnj: Use toString in toLiteral
62 ec7f10d 2016-12-17T14:38:30-05:00 GitHub: Merge pull request #102 from ExtendLang/
    circular-hotfix
63 73c454b 2016-12-17T14:34:42-05:00 GitHub: Merge branch 'master' into circular-hotfix
64 8126e2e 2016-12-17T14:24:22-05:00 oracleofnj: native toString
65 037728d 2016-12-17T13:46:34-05:00 Nigel Schuster: A lot of wrong paths make it work
66 0a4fd9d 2016-12-17T13:26:34-05:00 Nigel Schuster: Next attempt
67 56905f8 2016-12-17T13:19:59-05:00 Nigel Schuster: Merge branch 'plotting' of https://
    github.com/ExtendLang/Extend into plotting
68 fbf3a1e 2016-12-17T13:19:52-05:00 Nigel Schuster: Manual install (maybe?)
69 1171b71 2016-12-17T13:10:19-05:00 GitHub: Merge branch 'master' into plotting
70 0dbf85d 2016-12-17T13:07:56-05:00 Nigel Schuster: Added libgd for travis
71 060ae45 2016-12-17T13:01:35-05:00 oracleofnj: Merge branch 'master' into stdlib-string
72 4402208 2016-12-17T13:01:24-05:00 oracleofnj: Add round
73 23c2ae6 2016-12-17T13:00:05-05:00 GitHub: Merge pull request #123 from ExtendLang/size
    -asserts
74 0ef936b 2016-12-17T12:33:31-05:00 oracleofnj: Fix merge conflicts
75 4c51203 2016-12-17T11:59:30-05:00 oracleofnj: Right confusion
76 c05cf61 2016-12-17T11:52:34-05:00 oracleofnj: Fix import dir bug
77 39edbb4 2016-12-17T11:46:06-05:00 oracleofnj: Merge branch 'master' into size-asserts
78 339cb1f 2016-12-17T11:45:54-05:00 oracleofnj: Fix merge conflict
79 7462381 2016-12-17T11:44:50-05:00 GitHub: Merge pull request #122 from ExtendLang/
    split-stdlib
80 61ac8f2 2016-12-17T11:38:19-05:00 oracleofnj: Size asserts

```

```

81 606af9f 2016-12-17T11:22:36-05:00 oracleofnj: Transform asserts into more useful form;
    add calc of assert value to codegen
82 8743e4c 2016-12-17T11:07:31-05:00 Nigel Schuster: Explicit maxfloat
83 0f96e70 2016-12-17T11:02:09-05:00 Nigel Schuster: merge master; Keep tc around for
    testing
84 1882524 2016-12-17T10:54:10-05:00 Nigel Schuster: Creating archive
85 ee4f369 2016-12-17T10:40:17-05:00 oracleofnj: Combine asserts into a single expression
86 0f0f1c8 2016-12-17T10:38:56-05:00 Nigel Schuster: Added right and left to stdlib
87 4dc1597 2016-12-17T10:35:11-05:00 Nigel Schuster: Made compiling workable
88 fa43425 2016-12-17T10:30:23-05:00 oracleofnj: Split stdlib
89 824c53c 2016-12-17T10:11:13-05:00 Nigel Schuster: Added toUpper and toLower
90 ec24177 2016-12-17T10:02:30-05:00 Nigel Schuster: Implemented to and from ASCII
91 ab2e8f8 2016-12-17T09:15:39-05:00 GitHub: Merge pull request #116 from ExtendLang/line-
    -plus
92 5d1610b 2016-12-17T09:08:57-05:00 GitHub: Merge branch 'master' into line-plus
93 df3a827 2016-12-17T09:08:48-05:00 GitHub: Merge pull request #117 from ExtendLang/cmd-
    args
94 32a3487 2016-12-17T09:02:27-05:00 GitHub: Merge branch 'master' into cmd-args
95 a8f9d33 2016-12-17T09:00:23-05:00 Nigel Schuster: Args
96 bfccf0c 2016-12-17T08:58:08-05:00 Nigel Schuster: Cut down line count for plus
97 a6bc89a 2016-12-17T08:48:31-05:00 GitHub: Merge pull request #114 from ExtendLang/only-
    -new-string
98 5c96b7f 2016-12-17T08:03:27-05:00 GitHub: Merge pull request #109 from ExtendLang/unop-
    -bitnot
99 3834210 2016-12-17T00:33:37-05:00 oracleofnj: Get rid of box string in favor of
    new_string_all_the_way, renamed new_string
100 375bea7 2016-12-16T23:56:35-05:00 oracleofnj: Merge branch 'unop-bitnot' into remove-
    interpreter
101 fb1bd77 2016-12-16T23:54:43-05:00 oracleofnj: Clean up; remove interpreter; change
    DimInt to DimOneByOne
102 539dd75 2016-12-16T23:46:35-05:00 GitHub: Merge branch 'master' into unop-bitnot
103 5668e53 2016-12-16T23:43:57-05:00 Nigel Schuster: Using lrint instead of fptosi
104 45691eb 2016-12-16T23:35:38-05:00 GitHub: Merge pull request #111 from ExtendLang/
    global-semant
105 2cdfb8b 2016-12-16T23:33:26-05:00 GitHub: Merge branch 'master' into global-semant
106 c9500d9 2016-12-16T23:33:14-05:00 GitHub: Merge pull request #112 from ExtendLang/
    remove-function-signatures
107 0c24f54 2016-12-16T23:25:23-05:00 oracleofnj: Remove return signature from grammar and
    all test cases
108 e7f2864 2016-12-16T23:03:53-05:00 oracleofnj: Merge branch 'cleanup-1' into global-
    semant
109 567507e 2016-12-16T22:53:20-05:00 oracleofnj: Check globals; use same symbol_table
    function for semant and codegen
110 33e3942 2016-12-16T22:11:13-05:00 GitHub: Merge branch 'master' into plotting
111 55d8185 2016-12-16T22:00:30-05:00 Nigel Schuster: Removed comments and unnecessary
    files
112 629042f 2016-12-16T21:37:07-05:00 GitHub: Merge branch 'master' into unop-bitnot
113 48b139a 2016-12-16T21:34:09-05:00 Nigel Schuster: Implemented unary bitnot
114 39b02cd 2016-12-16T21:27:22-05:00 oracleofnj: Merge branch 'master' into global-semant
115 28c0983 2016-12-16T21:27:05-05:00 oracleofnj: Remove leftover printf
116 dc182df 2016-12-16T21:09:00-05:00 GitHub: Merge pull request #105 from ExtendLang/rg-
    eq
117 8cdf5c4 2016-12-16T19:31:26-05:00 oracleofnj: Expand test cases for range equality
118 41a3ccc 2016-12-16T19:18:44-05:00 GitHub: Merge branch 'master' into rg-eq
119 8dbebc1 2016-12-16T19:18:15-05:00 GitHub: Merge pull request #104 from ExtendLang/
    prevent-overlapping-formulas

```

```

120 c1431b5 2016-12-16T18:55:07-05:00 Nigel Schuster: Implemented basic subrange
    comparison
121 546536e 2016-12-16T18:47:12-05:00 oracleofnj: Detect overlapping formulas and give
    runtime error if present
122 3562e1b 2016-12-16T18:45:12-05:00 oracleofnj: Merge branch 'sr-val-fix' into prevent-
    overlapping-formulas
123 8713fa0 2016-12-16T18:42:40-05:00 oracleofnj: Checking
124 77d80b9 2016-12-16T18:26:31-05:00 Nigel Schuster: Fixed check for subrange
125 69fb0d2 2016-12-16T17:46:48-05:00 oracleofnj: Circular hotfix
126 4a3ec8d 2016-12-16T17:21:18-05:00 oracleofnj: Add concat
127 962c744 2016-12-16T12:09:00-05:00 GitHub: Merge pull request #101 from ExtendLang/
    finishing-these-range-literals
128 f234e00 2016-12-16T00:21:06-05:00 oracleofnj: Merge branch 'more-stdlib-functions'
    into finishing-these-range-literals
129 c9246ce 2016-12-16T00:20:59-05:00 oracleofnj: testing testing
130 6914039 2016-12-16T00:14:09-05:00 oracleofnj: Third time's the charm
131 4617e44 2016-12-16T00:01:12-05:00 oracleofnj: It compiles now
132 1d8e290 2016-12-15T23:42:43-05:00 oracleofnj: Fingers crossed
133 c9d28d3 2016-12-15T21:50:01-05:00 oracleofnj: Move all initializations into their own
    function; only box strings once
134 1cfd16 2016-12-15T18:47:30-05:00 oracleofnj: Merge branch 'master' into more-stdlib-
    functions
135 19c2beb 2016-12-15T18:40:12-05:00 oracleofnj: Try a couple more things out
136 845cb04 2016-12-15T18:33:07-05:00 GitHub: Merge pull request #96 from ExtendLang/
    ternary-fix
137 4bfb3bc 2016-12-15T18:23:00-05:00 oracleofnj: Merge branch 'ternary-fix' into more-
    stdlib-functions
138 ae55ca4 2016-12-15T18:21:58-05:00 oracleofnj: Define cell_row, cell_col
139 30a5db6 2016-12-15T18:19:56-05:00 oracleofnj: Merge branch 'ternary-fix' into more-
    stdlib-functions
140 b9f1f10 2016-12-15T18:17:53-05:00 oracleofnj: What is truth?
141 ac84c2f 2016-12-15T18:15:37-05:00 oracleofnj: Fix ternary to work properly with ranges
142 1f57d91 2016-12-15T17:03:26-05:00 oracleofnj: Look at this one
143 437ba46 2016-12-15T16:56:04-05:00 oracleofnj: Try this one
144 f0edf5b 2016-12-15T16:46:52-05:00 oracleofnj: Fixing bug
145 5ba31e6 2016-12-15T14:17:52-05:00 GitHub: Merge pull request #94 from ExtendLang/nan-
    inf
146 67c5739 2016-12-15T14:17:46-05:00 GitHub: Merge pull request #93 from ExtendLang/type-
    typeof
147 48a3d5c 2016-12-15T14:05:37-05:00 oracleofnj: Improve test case
148 8f08227 2016-12-15T13:58:46-05:00 oracleofnj: Add isNaN and isInfinite to stdlib
149 cbeec74 2016-12-15T13:30:31-05:00 oracleofnj: Rename token
150 9582228 2016-12-15T13:18:09-05:00 oracleofnj: Rename type to typeof
151 d1422c7 2016-12-15T10:42:19-05:00 GitHub: Merge pull request #92 from ExtendLang/
    compiler
152 66689bb 2016-12-15T09:08:56-05:00 Nigel Schuster: added working directory option,
    doing testing completely in tmp
153 a13ae93 2016-12-15T09:08:31-05:00 GitHub: Merge pull request #91 from ExtendLang/
    sizeof
154 a31add9 2016-12-15T09:08:13-05:00 GitHub: Merge pull request #90 from ExtendLang/
    subselect-C-side
155 2e67e06 2016-12-15T09:01:06-05:00 Nigel Schuster: Added option to specify compiler,
    using clang
156 c171450 2016-12-15T02:33:48-05:00 oracleofnj: SizeOf
157 c168044 2016-12-15T00:48:35-05:00 oracleofnj: Add row(), column() to codegen, add
    print_endline() to stdlib.xtnd

```

```

158 bf9426d 2016-12-15T00:27:13-05:00 oracleofnj: Print subrange
159 407ce41 2016-12-14T23:02:02-05:00 oracleofnj: Merge in subrange_string
160 756ea8e 2016-12-14T22:51:00-05:00 oracleofnj: Ranges
161 27a8e79 2016-12-14T22:16:13-05:00 oracleofnj: Resolve RHS slice
162 876d056 2016-12-14T22:02:56-05:00 oracleofnj: Resolve RHS index
163 b59e022 2016-12-14T21:46:00-05:00 Nigel Schuster: Added method to print subragne as
    string
164 a7d53a8 2016-12-14T19:55:38-05:00 oracleofnj: Merge branch 'master' into subselect-C-
    side
165 362e85b 2016-12-14T19:55:23-05:00 GitHub: Merge pull request #88 from ExtendLang/
    subselect
166 4912fa3 2016-12-14T19:40:10-05:00 oracleofnj: Add debug print info for slice
    structures
167 c1b33f4 2016-12-14T18:58:45-05:00 oracleofnj: Builder to end all builders
168 5d400c2 2016-12-14T18:55:06-05:00 oracleofnj: Add selection builders
169 29f6e28 2016-12-14T18:20:51-05:00 oracleofnj: Make additional infix operator for
    populating structure element
170 046d096 2016-12-14T17:49:19-05:00 oracleofnj: Set up RHS slice types
171 0d20933 2016-12-14T17:28:38-05:00 GitHub: Merge branch 'master' into plotting
172 614d84f 2016-12-14T17:25:20-05:00 Nigel Schuster: Dummy commit for travis
173 0e78574 2016-12-14T17:24:04-05:00 Nigel Schuster: Merge branch 'plotting' of https://
    github.com/ExtendLang/Extend into plotting
174 2da0d7d 2016-12-14T17:23:56-05:00 Nigel Schuster: Spelling fix
175 b25c2f5 2016-12-14T16:49:17-05:00 GitHub: Merge pull request #87 from ExtendLang/make-
    a-selection
176 7a12082 2016-12-14T16:43:38-05:00 oracleofnj: Move selection test cases back into
    inputs
177 e2c08d5 2016-12-14T16:31:00-05:00 oracleofnj: Make IDs work with deref_subrange
178 02f2f0c 2016-12-14T15:21:31-05:00 GitHub: Merge pull request #86 from ExtendLang/
    include-stdlib
179 8b0503f 2016-12-14T15:18:14-05:00 GitHub: Merge branch 'master' into include-stdlib
180 1f034a0 2016-12-14T15:17:52-05:00 GitHub: Merge pull request #84 from ExtendLang/math-
    linker
181 1e6dd91 2016-12-14T14:58:44-05:00 oracleofnj: Add expected output for slurp
182 ff1a5e3 2016-12-14T14:53:38-05:00 oracleofnj: Remove extend_ prefix from all sample
    code
183 81a2828 2016-12-14T14:48:38-05:00 oracleofnj: Automatically add extend_ prefix to
    external functions
184 dccled3 2016-12-14T14:30:52-05:00 oracleofnj: Fix samples
185 9b2c28f 2016-12-14T12:39:45-05:00 oracleofnj: Include stdlib automatically
186 13650ce 2016-12-14T12:35:21-05:00 Nigel Schuster: Merge branch 'math-linker' of https
    ://github.com/ExtendLang/Extend into math-linker
187 2e0d90d 2016-12-14T12:35:06-05:00 Nigel Schuster: Merge branch 'math-linker' of https
    ://github.com/ExtendLang/Extend into math-linker
188 83c689e 2016-12-14T12:34:14-05:00 Nigel Schuster: Merge branch 'math-linker' of https
    ://github.com/ExtendLang/Extend into math-linker
189 127f600 2016-12-14T12:34:07-05:00 Nigel Schuster: Include sys/resources
190 b34d97a 2016-12-14T12:03:44-05:00 GitHub: Merge branch 'master' into math-linker
191 8297f33 2016-12-14T12:01:47-05:00 GitHub: Merge pull request #85 from ExtendLang/put-
    lt-back
192 6b0c74f 2016-12-14T11:33:45-05:00 Nigel Schuster: Include sys/resources
193 37470e9 2016-12-14T11:14:06-05:00 oracleofnj: Put back LT, comment out sys/time.h
194 6bde590 2016-12-14T11:12:16-05:00 Nigel Schuster: Increasing stack size
195 6acc621 2016-12-14T11:03:31-05:00 Nigel Schuster: Disabled linking math when creating
    an intermediate
196 d87b73c 2016-12-14T10:51:58-05:00 GitHub: Merge pull request #82 from ExtendLang/hard-

```

```

to-repro-bug
197 d126e3c 2016-12-14T00:51:00-05:00 oracleofnj: Try with time.h instead of sys/time.h
198 a535612 2016-12-14T00:48:35-05:00 oracleofnj: Remove lrints
199 e844853 2016-12-14T00:34:37-05:00 oracleofnj: Initialize all variables and remove
    pointer math; bug appears fixed
200 4c1a421 2016-12-13T22:55:07-05:00 oracleofnj: Some formula is weird
201 5dbd409 2016-12-13T22:43:19-05:00 oracleofnj: Merge branch 'hard-to-repro-bug' of
    https://github.com/ExtendLang/Extend into hard-to-repro-bug
202 879eaf3 2016-12-13T22:43:17-05:00 oracleofnj: Testing
203 37f5ce2 2016-12-13T22:42:40-05:00 GitHub: Merge pull request #83 from ExtendLang/
    rounding-for-read
204 a1cfc5a 2016-12-13T22:34:21-05:00 Nigel Schuster: Added rounding at several places
205 e20f7e4 2016-12-13T21:36:13-05:00 oracleofnj: Half the time it works
206 9f97b1a 2016-12-13T20:38:08-05:00 GitHub: Merge branch 'master' into plotting
207 61bc9b6 2016-12-13T20:33:27-05:00 GitHub: Merge pull request #81 from ExtendLang/fix-
    em-all
208 4a810df 2016-12-13T19:34:29-05:00 Nigel Schuster: Corrected testcase outputs
209 ae5b8a8 2016-12-13T19:08:43-05:00 GitHub: Merge pull request #80 from ExtendLang/
    select
210 70b2704 2016-12-13T19:02:32-05:00 oracleofnj: No C99
211 15fd762 2016-12-13T18:42:21-05:00 oracleofnj: Merge branch 'master' into select
212 8e6e9ba 2016-12-13T18:42:05-05:00 GitHub: Merge pull request #78 from ExtendLang/unop-
    unary-minus
213 7a93885 2016-12-13T18:41:49-05:00 oracleofnj: Calculate all formula indices
214 07e63dc 2016-12-13T18:19:58-05:00 oracleofnj: Properly build instantiate var
215 1a29129 2016-12-13T17:24:16-05:00 oracleofnj: Replace bools with chars for
    compatibility between C and LLVM
216 12e78a3 2016-12-13T17:17:54-05:00 oracleofnj: Added debug output
217 a483282 2016-12-13T16:13:30-05:00 oracleofnj: Merge branch 'master' into unop-unary-
    minus
218 f8c9b43 2016-12-13T16:13:09-05:00 oracleofnj: Make TypeOf work
219 8146d04 2016-12-13T16:12:17-05:00 GitHub: Merge pull request #75 from ExtendLang/fix-
    more-tc
220 94afc93 2016-12-13T16:02:35-05:00 Nigel Schuster: Corrected expected TC
221 f6f8276 2016-12-13T16:00:59-05:00 Nigel Schuster: Fixed string.xtnd file
222 dcd5766 2016-12-13T15:44:38-05:00 GitHub: Merge pull request #74 from ExtendLang/fix-
    tc
223 bfe1c07 2016-12-13T15:39:45-05:00 oracleofnj: Merge branch 'master' into unop-unary-
    minus
224 d9abfc0 2016-12-13T15:38:38-05:00 GitHub: Merge branch 'master' into fix-tc
225 50ed49c 2016-12-13T15:38:04-05:00 oracleofnj: Merging in main
226 23328f1 2016-12-13T15:37:18-05:00 GitHub: Merge pull request #73 from ExtendLang/and-
    or-xor
227 324779a 2016-12-13T15:32:26-05:00 Nigel Schuster: Corrected expected value
228 fafe2e6 2016-12-13T15:29:21-05:00 Nigel Schuster: Fixed string tc
229 022f05c 2016-12-13T15:23:59-05:00 Nigel Schuster: Fixed testcase
230 b12fe37 2016-12-13T15:18:57-05:00 Nigel Schuster: Implemented and, or and xor
231 90cbaa0 2016-12-13T15:16:31-05:00 Nigel Schuster: Added left and right shift
232 571ee7e 2016-12-13T14:56:05-05:00 Nigel Schuster: Merge branch 'power' of https://
    github.com/ExtendLang/Extend into power
233 aeab40d 2016-12-13T14:55:57-05:00 Nigel Schuster: Removed unnecessary level of
    indirection
234 e377567 2016-12-13T14:53:28-05:00 GitHub: Merge branch 'master' into power
235 6ad8512 2016-12-13T14:53:11-05:00 GitHub: Merge pull request #69 from ExtendLang/unop-
    unary-minus
236 71f395d 2016-12-13T14:46:27-05:00 Nigel Schuster: Power to the people of Extend

```

```

237 6a04209 2016-12-13T14:45:46-05:00 oracleofnj: Fix merge conflict
238 edb0ecc 2016-12-13T14:43:32-05:00 oracleofnj: Add unary minus
239 668a0eb 2016-12-13T14:37:19-05:00 GitHub: Merge pull request #68 from ExtendLang/mod-
    div
240 866b68f 2016-12-13T14:32:18-05:00 Nigel Schuster: Added modulo and division operation
241 46d5aa6 2016-12-13T14:26:35-05:00 oracleofnj: Merge branch 'master' into unop-typeof
242 84dfc33 2016-12-13T14:26:25-05:00 Nigel Schuster: Crunched some code
243 76210eb 2016-12-13T14:26:18-05:00 oracleofnj: Start on it
244 f4d5a81 2016-12-13T14:22:12-05:00 Nigel Schuster: Merge branch 'master' into
    simplification
245 f873242 2016-12-13T14:21:26-05:00 GitHub: Merge pull request #65 from ExtendLang/
    subtraction
246 fc94112 2016-12-13T14:20:35-05:00 Nigel Schuster: Added multiplication
247 6c26c2c 2016-12-13T14:19:07-05:00 GitHub: Merge branch 'master' into subtraction
248 4afd78e 2016-12-13T14:18:55-05:00 GitHub: Merge pull request #64 from ExtendLang/
    refactor-boolean-binops
249 d4d4388 2016-12-13T14:15:58-05:00 GitHub: Merge branch 'master' into refactor-boolean-
    binops
250 bd90241 2016-12-13T14:14:17-05:00 GitHub: Merge branch 'master' into subtraction
251 4042259 2016-12-13T14:13:09-05:00 Nigel Schuster: Added subtraction
252 663f399 2016-12-13T14:12:57-05:00 oracleofnj: Remove wildcard from BinOp pattern match
253 82a3db2 2016-12-13T14:11:31-05:00 Nigel Schuster: Merge branch 'master' into
    subtraction
254 1bf6bed 2016-12-13T14:09:47-05:00 oracleofnj: Add TransformedAway exception for LogAnd
    and LogOr
255 c7d4162 2016-12-13T14:02:13-05:00 GitHub: Merge pull request #63 from ExtendLang/more-
    binops
256 952778e 2016-12-13T14:01:54-05:00 oracleofnj: Change Lt, Lte in grammar; implement GTE
257 97821c8 2016-12-13T13:47:52-05:00 oracleofnj: GT
258 1e1f973 2016-12-13T13:44:36-05:00 Nigel Schuster: Subtraction
259 e0a883a 2016-12-13T13:37:57-05:00 oracleofnj: Remove NotEq from AST since != is parsed
    to UnOp(LogNot, BinOp(Eq, ...))
260 cc40008 2016-12-13T12:49:33-05:00 GitHub: Merge pull request #60 from ExtendLang/
    addition2
261 7123ebc 2016-12-13T12:41:09-05:00 GitHub: Merge branch 'master' into addition2
262 a656f57 2016-12-13T12:38:12-05:00 GitHub: Merge pull request #61 from ExtendLang/debug
    -unop
263 c3a96a9 2016-12-13T12:37:31-05:00 Nigel Schuster: Merge branch 'master' into plotting
264 f59d962 2016-12-13T12:34:49-05:00 Nigel Schuster: Moved make of lib to travis script
265 eb134b3 2016-12-13T12:29:53-05:00 Nigel Schuster: Moved testcases
266 044c6bd 2016-12-13T12:29:07-05:00 Nigel Schuster: Fixed off by one error
267 a64cc15 2016-12-13T12:14:45-05:00 oracleofnj: Add Debug expr
268 59858a0 2016-12-13T11:33:12-05:00 oracleofnj: Whoops no space
269 0426f34 2016-12-13T11:30:26-05:00 oracleofnj: Add test case
270 49ffa86 2016-12-13T11:19:14-05:00 GitHub: Merge branch 'master' into addition2
271 81533f4 2016-12-13T11:13:44-05:00 GitHub: Merge pull request #59 from ExtendLang/equal
    -rights
272 3cdaa5a 2016-12-13T11:12:41-05:00 Nigel Schuster: String addition
273 64d1760 2016-12-13T11:04:55-05:00 oracleofnj: Wake up please, GitHub
274 840aeaf 2016-12-13T10:48:03-05:00 oracleofnj: Remove usage demonstration
275 61ff439 2016-12-13T03:26:35-05:00 oracleofnj: Add string equality and test cases
276 f3112e9 2016-12-13T01:57:10-05:00 oracleofnj: Reduce cut & paste
277 08ce677 2016-12-13T01:35:46-05:00 oracleofnj: Remove obsolete testing file
278 ae8a07e 2016-12-13T01:23:26-05:00 oracleofnj: Merge branch 'print_value_p' into equal-
    rights
279 6090713 2016-12-13T01:22:47-05:00 oracleofnj: Use correct printf specifier

```

```

280 862b38c 2016-12-13T01:19:14-05:00 oracleofnj: Merge branch 'print_value_p' into equal-
      rights
281 5e913ad 2016-12-13T01:16:07-05:00 oracleofnj: Add debug_print; remove print statement
      that was causing us to falsely pass test cases from to_string; show usage in UnOp(
      Neg)
282 50281b1 2016-12-13T00:47:28-05:00 oracleofnj: Numeric equality
283 0f76aa4 2016-12-12T22:30:15-05:00 oracleofnj: Remove print flags
284 200b8b6 2016-12-12T22:16:15-05:00 GitHub: Merge pull request #57 from ExtendLang/
      addition2
285 da7c543 2016-12-12T12:43:31-05:00 Nigel Schuster: Setting flag for addition
286 7e7276b 2016-12-12T12:37:35-05:00 Nigel Schuster: Merge branch 'master' into addition2
287 8834635 2016-12-12T10:18:51-05:00 GitHub: Merge pull request #55 from ExtendLang/
      runtime
288 53ae9e0 2016-12-12T10:06:24-05:00 GitHub: Merge branch 'master' into runtime
289 6ed303e 2016-12-12T09:43:57-05:00 GitHub: Merge pull request #56 from ExtendLang/
      truthy-fix
290 ae49ce6 2016-12-12T01:15:29-05:00 oracleofnj: Remove extra file
291 7fe6a22 2016-12-12T01:11:53-05:00 oracleofnj: Falsey fix
292 d1e196d 2016-12-12T00:23:13-05:00 Nigel Schuster: Extracted runtime into seperate file
293 ecc620e 2016-12-12T00:17:06-05:00 GitHub: Merge pull request #54 from ExtendLang/final
      -draft-for-real
294 4c8caa5 2016-12-12T00:09:16-05:00 GitHub: Merge branch 'master' into final-draft-for-
      real
295 04d3b57 2016-12-12T00:00:29-05:00 GitHub: Merge pull request #39 from ExtendLang/more-
      lrm-ed
296 39025b0 2016-12-11T23:59:18-05:00 Nigel Schuster: Fixed examples, made small
      corrections
297 a875b41 2016-12-11T23:51:30-05:00 GitHub: Merge pull request #53 from ExtendLang/
      truthy
298 616dd34 2016-12-11T23:15:54-05:00 oracleofnj: Merge branch 'master' into truthy
299 0fa8255 2016-12-11T23:14:42-05:00 oracleofnj: Apparently still needs some work
300 78584d7 2016-12-11T23:09:07-05:00 oracleofnj: Thanks a lot Travis
301 b5673d2 2016-12-11T22:51:52-05:00 oracleofnj: TERRRRRRRRR NARRRRRRR
      EEEEEEEEEEEEEEEEEEE
302 b81bc1b 2016-12-11T22:04:25-05:00 oracleofnj: Maybe Truthy
303 b95d14f 2016-12-11T21:02:28-05:00 GitHub: Merge pull request #50 from ExtendLang/
      builder-hotfix
304 6dea96f 2016-12-11T20:40:47-05:00 oracleofnj: So many builders
305 8aa125f 2016-12-11T20:15:52-05:00 Nigel Schuster: Made som rpgroess
306 2a905c7 2016-12-11T19:15:47-05:00 GitHub: Merge pull request #47 from ExtendLang/
      function-parameter
307 2bc6c85 2016-12-11T19:11:33-05:00 oracleofnj: Add combined test case
308 860a11b 2016-12-11T19:04:35-05:00 oracleofnj: Merge branch 'master' into function-
      parameter
309 8c3499e 2016-12-11T19:03:39-05:00 oracleofnj: Remove extraneous printlines
310 99418c0 2016-12-11T19:02:31-05:00 oracleofnj: Make function parameters work
311 6c00a72 2016-12-11T18:45:46-05:00 Nigel Schuster: Some progress
312 387559b 2016-12-11T18:39:00-05:00 oracleofnj: First attempt
313 18fc1be 2016-12-11T18:08:11-05:00 GitHub: Merge pull request #45 from ExtendLang/empty
314 d7590da 2016-12-11T17:42:46-05:00 GitHub: Merge branch 'master' into plotting
315 f7e9be8 2016-12-11T16:30:05-05:00 GitHub: Merge branch 'master' into empty
316 f1dd8a5 2016-12-11T16:18:44-05:00 GitHub: Merge pull request #46 from ExtendLang/
      actually-make-global-scope
317 50366f4 2016-12-11T15:38:05-05:00 oracleofnj: Make sure locals are properly masking
      globals
318 046c7cc 2016-12-11T15:30:53-05:00 oracleofnj: Make globals work, fix bug

```

```

319 a844a46 2016-12-11T15:14:09-05:00 oracleofnj: So close
320 18db166 2016-12-11T15:05:42-05:00 GitHub: Merge branch 'master' into empty
321 67849f0 2016-12-11T15:01:52-05:00 oracleofnj: Make the global scope object
322 393d02c 2016-12-11T14:25:02-05:00 Nigel Schuster: Implemented empty, small flag
    setting fix
323 3c4681d 2016-12-11T13:31:12-05:00 GitHub: Merge pull request #44 from ExtendLang/float
    -display-hotfix
324 7be1001 2016-12-11T13:26:55-05:00 GitHub: Merge branch 'master' into float-display-
    hotfix
325 b192a23 2016-12-11T13:26:48-05:00 Nigel Schuster: Added gdchart compile step
326 abcffd0 2016-12-11T13:19:05-05:00 GitHub: Merge pull request #42 from ExtendLang/
    encapsulate-build-scope
327 556da44 2016-12-11T13:18:15-05:00 oracleofnj: Floating point math hotfix
328 0ad195e 2016-12-11T12:42:42-05:00 oracleofnj: Merge branch 'master' into encapsulate-
    build-scope
329 9caf464 2016-12-11T12:41:40-05:00 oracleofnj: Encapsulate a little more of building
    the scope
330 1ae8d43 2016-12-11T12:23:04-05:00 Ishaan: Add new gitignore
331 6278c7b 2016-12-11T12:18:49-05:00 Ishaan: Rebase and add gdchart in lib/
332 5594687 2016-12-11T12:13:20-05:00 Ishaan: Remove images from version control
333 294a6db 2016-12-11T12:13:20-05:00 Ishaan: Write to file instead of stdout
334 08e9f75 2016-12-11T12:11:13-05:00 Ishaan: Add hardcoded graph functionality
335 d65aad4 2016-12-11T12:09:28-05:00 GitHub: Merge pull request #40 from ExtendLang/make-
    global-scope
336 b5b33f1 2016-12-11T12:09:12-05:00 Ishaan: Update gitignore to avoid the gdchart
    package
337 6746e8a 2016-12-11T12:09:12-05:00 Ishaan: Checking gif
338 83c2e09 2016-12-11T12:09:12-05:00 Ishaan: Add hardcoded plot function without params
    or installation
339 0f5a6ba 2016-12-11T12:04:05-05:00 oracleofnj: Merge branch 'master' into make-global-
    scope
340 56b58d9 2016-12-11T12:01:28-05:00 oracleofnj: Encapsulate build_var_defns
341 f25e5b3 2016-12-11T11:43:19-05:00 oracleofnj: Only construct var_defns once
342 9cee2fc 2016-12-11T10:07:36-05:00 Nigel Schuster: Testcases (#38)
343 f3f4bef 2016-12-11T00:45:44-05:00 oracleofnj: Make global variable to hold vardefns
344 a0ed757 2016-12-10T23:31:38-05:00 Nigel Schuster: Edited explanation for row() and
    column()
345 7c50ef2 2016-12-10T23:27:07-05:00 Nigel Schuster: Added info for strings
346 738e41b 2016-12-10T23:24:20-05:00 Nigel Schuster: Added boolean example
347 5377fdf 2016-12-10T23:19:26-05:00 Nigel Schuster: Added arithmetic example
348 a8f4ad9 2016-12-10T21:28:18-05:00 oracleofnj: Isolate the part of building a scope for
    reuse with global variables
349 58f7a4d 2016-12-10T18:05:01-05:00 Nigel Schuster: Performing copy before returning, so
    that memory can be freed with alloca
350 c0e56aa 2016-12-10T17:07:00-05:00 GitHub: Merge pull request #37 from ExtendLang/
    dereference
351 a4b35df 2016-12-10T16:42:17-05:00 Nigel Schuster: Removed obsolete methods
352 cf08a8c 2016-12-10T16:36:20-05:00 GitHub: Merge branch 'master' into dereference
353 ef0e5e7 2016-12-10T16:36:03-05:00 GitHub: Merge pull request #36 from ExtendLang/comp-
    warn
354 0177dc2 2016-12-10T16:35:50-05:00 GitHub: Merge pull request #35 from ExtendLang/
    linker
355 127f99d 2016-12-10T16:35:41-05:00 GitHub: Merge pull request #34 from ExtendLang/rel-
    import
356 b2e881d 2016-12-10T16:35:31-05:00 GitHub: Merge pull request #33 from ExtendLang/ts-
    fix

```



```

357 ce833d4 2016-12-10T16:14:34-05:00 Nigel Schuster: Dereferencing 1x1 subrange
358 e259556 2016-12-10T13:53:12-05:00 Nigel Schuster: Removed nodefaultlibs directive
359 09c3961 2016-12-10T13:50:19-05:00 Nigel Schuster: Modified linker to work for travis
360 36d662a 2016-12-10T13:37:27-05:00 Nigel Schuster: Attempt to link math
361 2d4564a 2016-12-10T13:22:14-05:00 Nigel Schuster: Linking math library
362 38ba6e6 2016-12-10T13:18:39-05:00 Nigel Schuster: Suppressing compiler warnings
363 9deac9b 2016-12-10T13:06:39-05:00 Nigel Schuster: Modified compile script. Removed
    debug output
364 d35607b 2016-12-10T13:04:30-05:00 Nigel Schuster: Simpler testscript
365 d37dac2 2016-12-10T12:36:45-05:00 Nigel Schuster: Fixed duplicate import issue
366 31c26bc 2016-12-10T12:30:29-05:00 Nigel Schuster: Added cmd args to link file
367 a350720 2016-12-10T11:40:50-05:00 Nigel Schuster: Switched import style from root
    directory to relative path
368 90e39b0 2016-12-10T11:24:19-05:00 Nigel Schuster: Fixed issue in testscript that might
    report false results when it fails early
369 718ecd3 2016-12-10T03:09:18-05:00 oracleofnj: Some changes to LRM; add if(a,b,c)
370 6a8f836 2016-12-09T18:29:22-05:00 GitHub: Merge pull request #24 from ExtendLang/final
    -draft-lrm
371 fc886a9 2016-12-09T18:23:52-05:00 oracleofnj: Merge branch 'final-draft-lrm'
372 cda63cb 2016-12-09T18:23:24-05:00 oracleofnj: Fix merge conflict
373 eac9e77 2016-12-09T18:04:08-05:00 GitHub: Merge pull request #29 from ExtendLang/
    refactor
374 fe825f4 2016-12-09T17:55:39-05:00 oracleofnj: Compact last bit
375 b02dbbe 2016-12-09T17:49:00-05:00 oracleofnj: Give formula functions names
376 edd7aa4 2016-12-09T17:40:57-05:00 Nigel Schuster: Removed artificats
377 9b49e20 2016-12-09T17:37:59-05:00 Nigel Schuster: Fixed I/O testcases
378 a4ad4b1 2016-12-09T17:18:13-05:00 Nigel Schuster: Merge
379 b07398b 2016-12-09T17:17:19-05:00 Nigel Schuster: Added macro for function definition
380 ed01567 2016-12-09T17:17:06-05:00 oracleofnj: Make sizeof not break tests
381 a0a7054 2016-12-09T17:01:20-05:00 oracleofnj: Use symbol table
382 56fd61b 2016-12-09T16:11:10-05:00 oracleofnj: Merge branch 'refactor' of https://
    github.com/ExtendLang/Extend into refactor
383 38aedba 2016-12-09T16:10:35-05:00 oracleofnj: Create symbol table
384 dfb702e 2016-12-09T16:01:08-05:00 Nigel Schuster: Converted more to value_p from
    subrange_p
385 e963186 2016-12-09T15:42:35-05:00 Nigel Schuster: Made example TC work
386 eb76234 2016-12-09T11:14:58-05:00 Nigel Schuster: Made Hello World work again
387 08aeb70 2016-12-09T02:13:09-05:00 oracleofnj: Done for the night
388 cb39114 2016-12-09T01:35:36-05:00 oracleofnj: More refactoring
389 7974bbd 2016-12-08T23:53:31-05:00 oracleofnj: Banish the term extern
390 49af972 2016-12-08T23:45:30-05:00 oracleofnj: Add a couple comments
391 0fbf461 2016-12-08T21:52:24-05:00 oracleofnj: Get my bearings
392 5ecb599 2016-12-08T19:47:51-05:00 Nigel Schuster: Added some documentation
393 65066fc 2016-12-08T12:18:57-05:00 Nigel Schuster: Added name display for variable
394 fb18949 2016-12-07T23:44:17-05:00 oracleofnj: Merge branch 'master' into final-draft-
    lrm
395 4aab3dc 2016-12-07T23:43:25-05:00 oracleofnj: Update PDF
396 ed44d27 2016-12-07T23:43:01-05:00 oracleofnj: Fix failing test cases
397 9354fa7 2016-12-07T23:06:36-05:00 oracleofnj: Final draft candidate
398 78649f4 2016-12-07T18:09:46-05:00 oracleofnj: Almost done
399 05ded19 2016-12-07T15:47:52-05:00 oracleofnj: More work
400 f985cc8 2016-12-07T12:14:59-05:00 Nigel Schuster: Merge branch 'finish-transformations
    ' into get-val-rev
401 4b58ce9 2016-12-07T12:13:23-05:00 Nigel Schuster: Tried to add more instructions
402 0722412 2016-12-07T11:32:11-05:00 oracleofnj: Working
403 099efe7 2016-12-07T10:48:35-05:00 Nigel Schuster: Making progress on evaluating

```

```

dimensions
404 fa09df7 2016-12-07T09:51:23-05:00 Nigel Schuster: Finally it works
405 cbb0577 2016-12-07T02:35:06-05:00 oracleofnj: Still WIP
406 e3c9436 2016-12-07T00:44:22-05:00 oracleofnj: WIP
407 b265e74 2016-12-07T00:41:23-05:00 Nigel Schuster: test commit to look at
408 18bb182 2016-12-07T00:35:06-05:00 oracleofnj: Still work in progress
409 a4554c0 2016-12-06T23:14:32-05:00 Nigel Schuster: At least it compiles
410 3432484 2016-12-06T22:42:22-05:00 Nigel Schuster: Getting closer. Need to add var_defn
    wrapper in build_formula
411 05145ca 2016-12-06T21:10:11-05:00 Nigel Schuster: Minor fix
412 af69b92 2016-12-06T17:23:45-05:00 oracleofnj: More updates
413 a65c24e 2016-12-06T16:14:10-05:00 oracleofnj: Merge branch 'master' into finish-
    transformations
414 85a4ccb 2016-12-06T16:12:31-05:00 oracleofnj: LRM update part 1
415 174a7b8 2016-12-06T11:09:31-05:00 Nigel Schuster: Made partial progress on
    implementing variable instantiation and such
416 90fc58e 2016-12-05T22:14:41-05:00 GitHub: Merge pull request #23 from ExtendLang/read-
    empty
417 767851d 2016-12-05T16:18:17-05:00 Nigel Schuster: Finished C side implementation of
    getVal
418 6b837d4 2016-12-05T16:06:34-05:00 Nigel Schuster: Merge branch 'master' into get-val
419 04c2c65 2016-12-05T15:53:35-05:00 oracleofnj: Add slurp by passing 0 max bytes
420 d8cf316 2016-12-05T14:46:46-05:00 oracleofnj: Start handling empty
421 910bd01 2016-12-05T14:27:07-05:00 GitHub: Merge pull request #21 from ExtendLang/
    fileio
422 1ce7f83 2016-12-05T14:18:41-05:00 oracleofnj: Create patch file
423 88480fb 2016-12-05T13:36:28-05:00 GitHub: Merge branch 'master' into fileio
424 29d02d9 2016-12-05T13:34:27-05:00 oracleofnj: Fix merge conflict - keep expr_loc
425 52e7a8a 2016-12-05T13:32:54-05:00 GitHub: Merge pull request #22 from ExtendLang/rm-
    micro
426 bfa906b 2016-12-05T13:28:03-05:00 oracleofnj: Fix off-by-one bug
427 eb8dd71 2016-12-05T13:20:03-05:00 oracleofnj: Address issues
428 flbllee 2016-12-05T12:46:35-05:00 Nigel Schuster: Skeleton for get_val
429 e4e5e26 2016-12-05T09:25:17-05:00 Nigel Schuster: Removed microc reference
    implementation
430 270da2b 2016-12-05T02:40:59-05:00 GitHub: Merge branch 'master' into fileio
431 b928e98 2016-12-05T02:40:10-05:00 Ishaan: Remove bloat
432 894b511 2016-12-05T02:32:49-05:00 Ishaan: Added testcase
433 62b8e83 2016-12-05T02:30:16-05:00 Ishaan: Added fwrite implementation
434 77a23ae 2016-12-05T01:39:30-05:00 Ishaan: Added read
435 46e9b58 2016-12-05T00:07:16-05:00 Ishaan: Make refactoring changes and new helpers
436 a5b9066 2016-12-04T14:00:30-05:00 GitHub: Merge pull request #20 from ExtendLang/lhs-
    all-ids
437 35e9471 2016-12-04T13:38:44-05:00 oracleofnj: Put back Id(s) as it was
438 641d454 2016-12-04T13:36:36-05:00 oracleofnj: Always transform to ID on LHS, even for
    LitInts
439 0e8398f 2016-12-04T13:23:27-05:00 oracleofnj: Transform all LHS expressions including
    integers to IDs; check for strings or range literals and disallow
440 f47f2ba 2016-12-04T10:30:44-05:00 oracleofnj: Add error handling to close() and add a
    couple test cases
441 e95a95a 2016-12-04T10:07:01-05:00 oracleofnj: Add assertSingleNumber and get_number to
    eliminate more copy & paste
442 543e720 2016-12-04T09:47:03-05:00 oracleofnj: Add new_number() to eliminate some copy
    and paste
443 d7f10c9 2016-12-04T02:31:03-05:00 Ishaan: Tentative drafts of fileio functions
444 7d81e43 2016-12-04T00:15:20-05:00 oracleofnj: add diagnostic printf

```

445 868d9a4 2016-12-03T23:46:01-05:00 Ishaan: Cleanup  
 446 aa1e014 2016-12-03T23:42:46-05:00 Ishaan: Add file pointer array  
 447 88d05de 2016-12-03T18:38:34-05:00 Ishaan: Working on fopen  
 448 36f5848 2016-12-03T14:07:39-05:00 oracleofnj: Merge branch 'master' into finish-transformations  
 449 2ae2b83 2016-12-03T14:06:40-05:00 GitHub: Merge pull request #15 from ExtendLang/stdlib-fun  
 450 7c78a23 2016-12-03T14:02:51-05:00 oracleofnj: Move test\_fabs out of regression test suite  
 451 0a8055b 2016-12-03T13:48:19-05:00 oracleofnj: make test | grep REGRESSION  
 452 a24742b 2016-12-02T22:50:43-05:00 Kevin: Merged stdlib with master  
 453 5243c5a 2016-12-02T18:16:36-05:00 Kevin: Removed magic numbers and add fabs test  
 454 330bec3 2016-12-02T13:49:34-05:00 oracleofnj: Merge branch 'master' into finish-transformations  
 455 8a60995 2016-12-01T23:38:54-05:00 GitHub: Merge pull request #18 from ExtendLang/parser-error  
 456 f0d33e2 2016-12-01T23:18:39-05:00 oracleofnj: Move error handling  
 457 3b24c3a 2016-12-01T23:16:53-05:00 oracleofnj: Adjust test script  
 458 60a732f 2016-12-01T22:55:28-05:00 oracleofnj: Merge branch 'master' into parser-error  
 459 5dec6a2 2016-12-01T22:55:05-05:00 oracleofnj: Thank you Nigel!!!  
 460 96a3028 2016-12-01T22:19:21-05:00 GitHub: Merge pull request #16 from ExtendLang/fail-silent  
 461 6c3696c 2016-12-01T21:59:40-05:00 oracleofnj: Figure out why test is failing  
 462 7912d5a 2016-12-01T21:26:03-05:00 GitHub: Merge branch 'master' into fail-silent  
 463 9702e5b 2016-12-01T21:14:35-05:00 oracleofnj: Merge branch 'master' into finish-transformations  
 464 5bdd52c 2016-12-01T21:13:45-05:00 GitHub: Merge pull request #17 from ExtendLang/lexbuf-pos  
 465 8893255 2016-12-01T20:35:04-05:00 oracleofnj: Add a couple test cases  
 466 2868653 2016-12-01T20:23:01-05:00 oracleofnj: Use lexbuf.lex\_curr\_p to calculate position  
 467 8c7b6ce 2016-12-01T18:59:49-05:00 GitHub: Merge pull request #11 from ExtendLang/parse\_error  
 468 2885ac7 2016-12-01T18:56:15-05:00 Ishaan: Added test case for string  
 469 047cfec 2016-12-01T18:42:04-05:00 oracleofnj: Add short circuiting test cases  
 470 6acd7f6 2016-12-01T18:31:33-05:00 oracleofnj: Merge remote-tracking branch 'origin/fail-silent' into finish-transformations  
 471 72360f4 2016-12-01T17:09:08-05:00 Nigel Schuster: Minified error output for outputs that have not passed yet  
 472 5762112 2016-12-01T16:04:06-05:00 oracleofnj: Get rid of wildcard pattern match in interpreter  
 473 a90a343 2016-12-01T15:59:40-05:00 oracleofnj: Merge branch 'master' into finish-transformations  
 474 85bc21d 2016-12-01T15:59:05-05:00 oracleofnj: Remove unnecessary file  
 475 81fe565 2016-12-01T15:58:40-05:00 oracleofnj: Finish range literals  
 476 e9fb1c2 2016-12-01T15:04:03-05:00 Ishaan: Added increment to string buffer and tests  
 477 eb7cle8 2016-12-01T15:04:03-05:00 Ishaan: Add partial character indexing  
 478 df09aea 2016-12-01T15:04:03-05:00 Ishaan: Add expected parse testcase intermediate  
 479 712a710 2016-12-01T15:04:03-05:00 Ishaan: Added tentative scanner-level line number  
 480 bf4ee6c 2016-12-01T15:04:03-05:00 Ishaan: Added SyntaxError Exception at scan level  
 481 da41520 2016-12-01T14:54:21-05:00 oracleofnj: So close  
 482 7abb394 2016-12-01T14:07:58-05:00 GitHub: Merge pull request #14 from ExtendLang/sinner  
 483 e0b7fdb 2016-12-01T14:05:38-05:00 Nigel Schuster: Rename empty to new\_val  
 484 2cabadc 2016-12-01T11:58:03-05:00 oracleofnj: Merge branch 'master' into finish-transformations

```

485 6ea8cff 2016-12-01T10:10:26-05:00 Nigel Schuster: Using define instead of magic
    numbers
486 cd7d261 2016-12-01T10:07:10-05:00 Nigel Schuster: Merge branch 'master' into sinner
487 13cd317 2016-12-01T10:06:25-05:00 GitHub: Merge pull request #13 from ExtendLang/
    value_p
488 cf36f70 2016-12-01T09:47:38-05:00 oracleofnj: Sample digits function
489 4eed07 2016-12-01T01:02:56-05:00 Ishaan: Change print return type to empty
490 fa42f27 2016-12-01T00:41:47-05:00 Kevin: Fixed acos function
491 53d34ad 2016-12-01T00:29:32-05:00 Nigel Schuster: Moved double values type to numeric
492 f769c61 2016-12-01T00:18:07-05:00 Nigel Schuster: Merge branch 'sinner' into stdlib-
    fun
493 3986f38 2016-12-01T00:17:21-05:00 Nigel Schuster: Merge branch 'value_p' into sinner
494 5bd87f9 2016-12-01T00:14:45-05:00 Nigel Schuster: Explicitly declaring to link math
    library
495 4604545 2016-12-01T00:12:08-05:00 Nigel Schuster: Consistently using floats
496 38b9824 2016-11-30T23:46:14-05:00 Nigel Schuster: Merge branch 'value_p' into sinner
497 3303575 2016-11-30T23:45:25-05:00 Nigel Schuster: Explicitly declaring to link math
    library
498 31a74ec 2016-11-30T23:35:34-05:00 Nigel Schuster: Merge branch 'master' into value_p
499 7f0bc86 2016-11-30T23:04:34-05:00 Kevin: Finished remainder of stdlib
500 cd160df 2016-11-30T22:50:18-05:00 Kevin: Added more c functions to stdlib
501 e085977 2016-11-30T19:59:57-05:00 Nigel Schuster: Made sin function work
502 206ee5a 2016-11-30T19:07:28-05:00 Nigel Schuster: Moved all function signatures to
    value_p return value
503 effc20b 2016-11-30T18:45:52-05:00 GitHub: Merge pull request #12 from ExtendLang/easy-
    compile
504 3b6d7b7 2016-11-30T17:51:19-05:00 Nigel Schuster: Added script to compile and link
505 febcbf8 2016-11-30T15:54:45-05:00 oracleofnj: Add oddball formula test case and try
    out theory for range literal
506 4a1ff4f 2016-11-30T14:54:05-05:00 oracleofnj: Finish reducing Ternary to
    ReducedTernary
507 8f0a981 2016-11-30T12:35:43-05:00 oracleofnj: Working on reducing ternaries
508 d3c5812 2016-11-30T02:39:58-05:00 oracleofnj: Finish desugaring switch
509 0a22713 2016-11-30T00:09:10-05:00 oracleofnj: Getting ready to ternarize switch
510 84f016a 2016-11-29T21:54:15-05:00 oracleofnj: Fix bug in switch() with default case
511 d331b7a 2016-11-29T17:33:41-05:00 oracleofnj: Give desugaring variables easier-to-read
    names for debugging purposes
512 36f8de5 2016-11-29T16:14:46-05:00 oracleofnj: Missed one
513 d96da34 2016-11-29T16:13:21-05:00 oracleofnj: Transform &&, || into ternary
    expressions to support proper short-circuit evaluation
514 3a8efbc 2016-11-28T23:05:28-05:00 GitHub: Merge pull request #9 from ExtendLang/func-
    calls
515 7a2af49 2016-11-28T20:33:53-05:00 Nigel Schuster: Removed another ocaml 4.3 dep
516 468e79f 2016-11-28T19:50:53-05:00 Nigel Schuster: Added ocaml 4.3 as dep for travis (
    hopefully this works)
517 a408761 2016-11-28T19:35:49-05:00 Nigel Schuster: Fixed String.equal
518 90c3caf 2016-11-27T22:52:14-05:00 Nigel Schuster: Fixed interpreter for now
519 a18da78 2016-11-27T22:42:27-05:00 Nigel Schuster: Added accidentally created file
520 5647312 2016-11-27T22:41:22-05:00 Nigel Schuster: Made extern function calls work
521 872aa8c 2016-11-27T13:52:44-05:00 Nigel Schuster: Merge branch 'func-calls' of https
    ://github.com/ExtendLang/Extend into func-calls
522 26ef1cc 2016-11-27T13:51:06-05:00 Nigel Schuster: Merging list of functions
523 877336f 2016-11-27T12:15:11-05:00 GitHub: Merge branch 'master' into func-calls
524 5b3edb0 2016-11-27T12:14:43-05:00 GitHub: Merge pull request #8 from ExtendLang/stdlib
    -template
525 374273f 2016-11-27T12:13:52-05:00 Nigel Schuster: Function calls work now

```

526 952aab8 2016-11-27T09:54:12-05:00 Nigel Schuster: Merge extern  
527 ac6268f 2016-11-26T23:06:00-05:00 Nigel Schuster: Boxing ints, added unop sizeof,  
actually returning subrange not dummy object  
528 ca07be3 2016-11-26T21:27:19-05:00 Nigel Schuster: Unboxing hello world to and from  
subrange  
529 aef6c19 2016-11-26T16:55:48-05:00 Nigel Schuster: Made Hello World somewhat workable  
530 cfb637e 2016-11-25T18:27:37-05:00 Nigel Schuster: Fixed faulty setup on call  
531 ebf926a 2016-11-25T17:48:57-05:00 Nigel Schuster: Added template in C  
532 554fbb2 2016-11-23T22:28:29-05:00 oracleofnj: Better error message for WrongNumberArgs  
533 f09e40e 2016-11-23T12:47:39-05:00 oracleofnj: Make sequence work  
534 053980b 2016-11-22T16:02:27-05:00 oracleofnj: Actually commit all the extern stuff  
535 0e0fa23 2016-11-22T14:36:54-05:00 Nigel Schuster: Added extern in Ast  
536 aac63be 2016-11-21T23:52:25-05:00 oracleofnj: Better duplicate definition checking  
537 08e2d07 2016-11-21T23:29:28-05:00 oracleofnj: Check assertions before evaluating fn  
return expression  
538 69fa332 2016-11-21T18:01:23-05:00 oracleofnj: Add size assertions  
539 22541c4 2016-11-21T12:48:34-05:00 oracleofnj: Fix bug in Call()  
540 9a1d24b 2016-11-21T12:39:41-05:00 oracleofnj: Working on crazy bug  
541 a485cee 2016-11-20T22:13:46-05:00 oracleofnj: Add test case for foo([m, n] arg)  
542 10afe9a 2016-11-20T22:07:17-05:00 oracleofnj: Expand function signature  
543 325e9ba 2016-11-20T18:53:52-05:00 oracleofnj: Well, this is awkward  
544 0a76dc9 2016-11-20T18:41:12-05:00 oracleofnj: Add check of return value  
545 488e34e 2016-11-20T18:31:39-05:00 oracleofnj: Add sample #1  
546 93eebc5 2016-11-20T18:27:23-05:00 oracleofnj: Add semantic checking to make sure  
functions and variables on RHS exist  
547 881f164 2016-11-20T17:22:40-05:00 oracleofnj: Check RHS slice to ensure end > start,  
otherwise evaluate to empty  
548 442ae91 2016-11-20T11:42:54-05:00 GitHub: Merge pull request #73 from Neitsch/  
interpreter-global  
549 f7f701d 2016-11-20T11:30:06-05:00 Nigel Schuster: Added use of global variables to  
interpreter, fixed specs for logical or and and testcases with empty  
550 367bc2b 2016-11-20T00:33:17-05:00 GitHub: Merge pull request #72 from Neitsch/codegen-  
part-app-fix  
551 bdca834 2016-11-20T00:31:04-05:00 GitHub: Merge branch 'master' into codegen-part-app-  
fix  
552 e956238 2016-11-20T00:28:49-05:00 GitHub: Merge pull request #71 from Neitsch/tc-fixes  
553 9b742d1 2016-11-20T00:24:39-05:00 Nigel Schuster: Fixed partial function application  
warning  
554 32f2989 2016-11-20T00:20:51-05:00 GitHub: Merge branch 'master' into tc-fixes  
555 f87cb94 2016-11-20T00:20:35-05:00 GitHub: Merge pull request #69 from Neitsch/  
regression-tests  
556 842ee5a 2016-11-20T00:18:56-05:00 GitHub: Merge branch 'master' into regression-tests  
557 6d73717 2016-11-19T23:55:35-05:00 GitHub: Merge pull request #66 from Neitsch/fix-test-  
-cases  
558 05f317a 2016-11-19T22:37:36-05:00 Nigel Schuster: Fixed output on TCs  
559 aa1d974 2016-11-19T22:33:40-05:00 Nigel Schuster: Fixed expected value for ternary  
560 ab7653a 2016-11-19T22:32:27-05:00 Nigel Schuster: Fixed import testcases  
561 848066c 2016-11-19T22:24:55-05:00 Nigel Schuster: Moved testcase asset to asset folder  
562 53c9206 2016-11-19T22:21:48-05:00 Nigel Schuster: Corrected use of global variable in  
test\_globals  
563 5fe74a8 2016-11-19T22:21:00-05:00 Nigel Schuster: Fixed expected output for  
test\_access\_column\_cells  
564 214ab9d 2016-11-19T22:10:33-05:00 Nigel Schuster: Merge  
565 fb31505 2016-11-19T22:08:42-05:00 Nigel Schuster: Passing testcases are in separate  
directory. Output of stats  
566 5e39ba7 2016-11-19T21:55:03-05:00 Nigel Schuster: Merge

```

567 25263fe 2016-11-19T21:51:31-05:00 Nigel Schuster: Removed travis from build, removed
    super verbose output
568 0554ad9 2016-11-19T21:42:28-05:00 Nigel Schuster: Using precise lli version
569 04e5c4a 2016-11-19T18:30:32-05:00 oracleofnj: Add more operators to interpreter
570 e4a190c 2016-11-19T17:14:04-05:00 oracleofnj: Add argument to main and remove
    _expected from filenames
571 7cd2b3a 2016-11-19T16:53:12-05:00 oracleofnj: Merge branch 'master' into fix-test-
    cases
572 d1fddfd 2016-11-19T16:52:48-05:00 oracleofnj: Merge branch 'fix-test-cases' of https
    ://github.com/Neitsch/plt into fix-test-cases
573 36f72a1 2016-11-19T16:49:34-05:00 GitHub: Merge pull request #67 from Neitsch/
    test_cases
574 c46c87b 2016-11-19T16:47:26-05:00 GitHub: Merge branch 'master' into test_cases
575 642ce76 2016-11-19T16:39:50-05:00 Kevin: Fixed helloworld bug
576 ac3d7fa 2016-11-19T16:10:53-05:00 Kevin: Added corresponding AST result for gcd
    function
577 7b6b79e 2016-11-19T14:31:39-05:00 GitHub: Merge branch 'master' into fix-test-cases
578 a9320f3 2016-11-19T14:29:51-05:00 oracleofnj: Merge branch 'master' into fix-test-
    cases
579 24a3625 2016-11-19T14:27:48-05:00 oracleofnj: Add switch tests
580 de262b4 2016-11-19T14:24:39-05:00 GitHub: Merge pull request #60 from Neitsch/box-args
581 75e3f71 2016-11-18T20:39:23-05:00 oracleofnj: Fix parsing errors in test cases
582 4e38757 2016-11-18T16:00:10-05:00 GitHub: Merge branch 'master' into box-args
583 7146dce 2016-11-18T15:59:54-05:00 GitHub: Merge pull request #64 from Neitsch/reorg-
    test
584 f483ac7 2016-11-18T14:10:32-05:00 Kevin: Updated print statement for each test
585 09cb42f 2016-11-18T14:07:39-05:00 oracleofnj: Fix parse difference
586 39634bb 2016-11-18T14:01:21-05:00 oracleofnj: Remove unnecessary files
587 d772725 2016-11-18T14:01:02-05:00 oracleofnj: Make inputs work with interpreter
588 f4456f8 2016-11-18T13:17:25-05:00 GitHub: Merge branch 'master' into test_cases
589 00aafb7 2016-11-18T13:16:08-05:00 Kevin: Renamed inputs folder
590 99db652 2016-11-18T12:51:40-05:00 Kevin: Renamed expected output extension and created
    input folder for test cases
591 2825ada 2016-11-18T12:51:33-05:00 Nigel Schuster: Added branch to build
592 aafabb2 2016-11-18T12:50:56-05:00 Nigel Schuster: Verbose output for travis debug
593 124d61e 2016-11-18T12:44:50-05:00 GitHub: Merge pull request #61 from Neitsch/reorg-
    test
594 82cf599 2016-11-18T12:34:57-05:00 oracleofnj: Modify test script to compare
    interpreter and compiler with expected
595 faecfa1 2016-11-18T01:48:44-05:00 oracleofnj: Fix merge conflict in box_args
596 41a81ce 2016-11-18T01:40:11-05:00 oracleofnj: Move argument boxing into a function
597 6f63e89 2016-11-18T00:48:07-05:00 GitHub: Merge pull request #59 from Neitsch/hello-
    hello
598 088dc45 2016-11-18T00:29:45-05:00 Nigel Schuster: Merge
599 012caaa 2016-11-18T00:12:40-05:00 GitHub: Merge pull request #58 from Neitsch/copy-
    argv
600 f84757b 2016-11-18T00:02:34-05:00 Nigel Schuster: Removed unnecessary files
601 18fbff1 2016-11-18T00:01:49-05:00 Nigel Schuster: Removed dummy arg reading, added
    printing to interpreter - helloworld TC passes
602 b866da3 2016-11-17T23:31:42-05:00 Nigel Schuster: Made hello world work
603 9463afa 2016-11-17T23:12:41-05:00 oracleofnj: Merge branch 'copy-argv' of https://
    github.com/Neitsch/plt into copy-argv
604 54858ab 2016-11-17T23:11:29-05:00 oracleofnj: Add => infix operator to cut down on all
    the build_struct_gep calls
605 bb11d6d 2016-11-17T23:10:24-05:00 GitHub: Merge branch 'master' into copy-argv
606 e123652 2016-11-17T22:28:12-05:00 oracleofnj: Add byte for zero

```

```

607 26a03b7 2016-11-17T22:24:17-05:00 oracleofnj: Add new_string function
608 b8028f9 2016-11-17T20:27:37-05:00 Kevin: Removed files from test folder
609 c85d9b7 2016-11-17T20:25:21-05:00 Kevin: Move testcases to testcases directory
610 f17c6b6 2016-11-17T20:21:38-05:00 Kevin Ye: Complete testcases for List/Range/Function
    /Expression with expected outputs
611 5e63cee 2016-11-17T17:40:31-05:00 GitHub: Merge pull request #54 from Neitsch/
    operation_tests
612 4a4a806 2016-11-17T17:19:13-05:00 GitHub: Merge branch 'master' into operation_tests
613 cafe20e 2016-11-17T17:19:11-05:00 GitHub: Merge pull request #52 from Neitsch/one-main-
    -arg
614 4b28df2 2016-11-17T17:17:44-05:00 GitHub: Merge branch 'master' into operation_tests
615 b728e2e 2016-11-17T17:16:20-05:00 GitHub: Merge branch 'master' into one-main-arg
616 d43a87b 2016-11-17T17:15:28-05:00 GitHub: Merge pull request #55 from Neitsch/shell-
    fix
617 b1238a0 2016-11-17T17:08:56-05:00 Nigel Schuster: Shell is not my strength
618 a6cc0ea 2016-11-17T17:05:09-05:00 Nigel Schuster: Screw you bourne shell
619 51f6e67 2016-11-17T16:59:50-05:00 Nigel Schuster: Using bourne shell style redirection
    :
620 3255e1b 2016-11-17T16:38:53-05:00 Ishaan: Modify test suite specs
621 f0ab4d8 2016-11-17T16:38:53-05:00 Ishaan: Moved expected output text files to
    directory
622 06d330c 2016-11-17T16:38:53-05:00 Ishaan: 75% through operator cases
623 e490548 2016-11-17T15:50:35-05:00 GitHub: Merge branch 'master' into one-main-arg
624 a4cf367 2016-11-17T15:50:29-05:00 GitHub: Merge pull request #51 from Neitsch/test-
    script
625 79ee3de 2016-11-17T15:18:58-05:00 oracleofnj: Call main() with first argument <empty>
    in interpreter
626 c4f7437 2016-11-17T14:39:38-05:00 Nigel Schuster: Removed version specific lli
627 7b2236b 2016-11-17T14:35:55-05:00 Nigel Schuster: Fixed if no flag is given
628 e10f656 2016-11-17T14:24:20-05:00 Nigel Schuster: Outputting diff only if -p flag is
    given
629 2d29597 2016-11-17T14:19:30-05:00 Nigel Schuster: Added it as build target
630 7af929a 2016-11-17T14:12:19-05:00 GitHub: Merge pull request #50 from Neitsch/test-
    script
631 6ea43f6 2016-11-17T13:54:55-05:00 Nigel Schuster: Added more env variables to avoid
    copy paste
632 05f27a2 2016-11-17T12:45:11-05:00 Nigel Schuster: Made simple testscript
633 aca43c1 2016-11-17T11:08:11-05:00 Nigel Schuster: Removed accidentally added files
634 9228eac 2016-11-17T04:52:31-05:00 Kevin Ye: Test cases for List of Tests and Range/
    Function/Expression Tests
635 7feb392 2016-11-17T00:28:53-05:00 GitHub: Merge pull request #48 from Neitsch/
    testing_list
636 6e42afa 2016-11-17T00:27:13-05:00 GitHub: Merge branch 'master' into testing_list
637 e40734b 2016-11-16T23:25:01-05:00 Ishaan: Added more test scenarios
638 41ef578 2016-11-16T17:50:03-05:00 GitHub: Merge pull request #49 from Neitsch/consume-
    command-line-args
639 3cbf089 2016-11-16T17:45:58-05:00 oracleofnj: Fix merge conflict
640 1570836 2016-11-16T16:51:05-05:00 GitHub: Merge pull request #45 from Neitsch/doc
641 a8fbced 2016-11-16T16:38:49-05:00 Nigel Schuster: Fixed minor syntax error
642 c2f37c8 2016-11-16T16:30:43-05:00 Nigel Schuster: Merge
643 2fa73be 2016-11-16T16:05:37-05:00 oracleofnj: Set return code to length of argv[1]
644 bc21af6 2016-11-16T15:54:12-05:00 Ishaan: Added initial testing list
645 cd0d156 2016-11-16T15:50:39-05:00 oracleofnj: Start processing command line args
646 4a1fcac 2016-11-16T13:55:46-05:00 GitHub: Merge pull request #46 from Neitsch/number-
    type
647 flb481e 2016-11-16T11:04:44-05:00 Nigel Schuster: Added number type that defaults to

```

```

    int
648 8944b9a 2016-11-16T00:19:33-05:00 GitHub: Merge pull request #44 from Neitsch/fix-arg
649 92fb7a3 2016-11-15T23:57:37-05:00 Nigel Schuster: Added a little documentation
650 bcbde36 2016-11-15T23:49:07-05:00 GitHub: Merge branch 'master' into fix-arg
651 fa1741a 2016-11-15T23:03:23-05:00 GitHub: Merge pull request #43 from Neitsch/more-
    llvm-gen-js
652 57b2162 2016-11-15T22:39:38-05:00 Nigel Schuster: Using subranges instead of ranges
    everywhere
653 9407677 2016-11-15T22:31:03-05:00 oracleofnj: Add hash table for common functions and
    add dereference-the-range
654 46e1fd5 2016-11-15T21:38:51-05:00 oracleofnj: Eliminate some copy & paste
655 660c049 2016-11-15T20:54:33-05:00 GitHub: Merge pull request #42 from Neitsch/llvm-gen
656 25b23cd 2016-11-15T17:23:54-05:00 Nigel Schuster: Fixed column retrieval for 1x1
657 3f02203 2016-11-15T17:17:02-05:00 Nigel Schuster: Fixed tests
658 26b8fcf 2016-11-15T17:15:08-05:00 Nigel Schuster: Merge
659 e347a87 2016-11-15T17:12:26-05:00 Nigel Schuster: Using more generic flag for values
660 aed28b3 2016-11-15T17:08:07-05:00 oracleofnj: Add is_subrange_1x1
661 cf5cbf0 2016-11-15T14:51:40-05:00 oracleofnj: Merge branch 'llvm-gen' of https://
    github.com/Neitsch/plt into llvm-gen
662 c71d469 2016-11-15T14:51:19-05:00 oracleofnj: Replace String.equal with =
663 4b34abd 2016-11-15T14:41:37-05:00 GitHub: Merge branch 'master' into llvm-gen
664 a80a6d0 2016-11-15T14:41:07-05:00 oracleofnj: Add compile option to main
665 8ad5a19 2016-11-15T14:33:40-05:00 GitHub: Merge pull request #40 from Neitsch/
    interpreter
666 3f0362a 2016-11-15T14:28:44-05:00 GitHub: Merge branch 'master' into interpreter
667 c0c95a2 2016-11-15T14:16:13-05:00 Nigel Schuster: Merge
668 d5f4024 2016-11-15T13:44:44-05:00 Nigel Schuster: Moved failing TCs
669 42fd9ef 2016-11-15T12:21:57-05:00 oracleofnj: Fix bug in import
670 9c567c9 2016-11-15T11:11:30-05:00 Nigel Schuster: Working on imports, fixed most
    testcases
671 aa61ac9 2016-11-15T09:31:42-05:00 Nigel Schuster: Allocating scope object
672 cf1ebf9 2016-11-13T23:09:30-05:00 oracleofnj: Rewrite main to take options; fix bug
    where import didn't know about first filename
673 5749538 2016-11-13T21:59:28-05:00 Nigel Schuster: Added main function
674 d6daff3 2016-11-13T20:26:14-05:00 GitHub: Merge pull request #41 from Neitsch/
    LRM_String_Update
675 0a5d484 2016-11-13T18:45:29-05:00 oracleofnj: Revert "Generating function header"
676 6afe599 2016-11-13T18:44:58-05:00 Ishaan Kolluri: Added changes relating to strings.
677 137d7e2 2016-11-13T18:39:33-05:00 oracleofnj: Merge branch 'interpreter' of https://
    github.com/Neitsch/plt into interpreter
678 118bfc5 2016-11-13T18:38:34-05:00 oracleofnj: Allow single slice on RHS; make hashtag
    work
679 e376270 2016-11-13T17:55:41-05:00 Nigel Schuster: Added type arguments for functions
680 5cfb519 2016-11-13T17:26:23-05:00 Nigel Schuster: Set more types up
681 bfl8dbb 2016-11-13T15:30:35-05:00 Nigel Schuster: Merge branch 'interpreter' of https
    ://github.com/Neitsch/plt into interpreter
682 f83a0bc 2016-11-13T15:30:28-05:00 Nigel Schuster: Generating function header
683 3addcc8 2016-11-13T14:38:11-05:00 oracleofnj: Make size(expr) an operator instead of
    built-in function
684 9a74e14 2016-11-13T14:22:44-05:00 oracleofnj: Changing size() to be an operator
685 d6d2eaa 2016-11-13T00:08:41-05:00 oracleofnj: Add closure to interpreter_variable
686 64fba82 2016-11-12T22:38:39-05:00 oracleofnj: Added bsearch to show logic bug
687 66ffdb1 2016-11-12T19:21:07-05:00 oracleofnj: Add alpha version of function calls
688 376b29a 2016-11-12T17:17:23-05:00 oracleofnj: Add string as value type
689 08c61ee 2016-11-12T17:14:47-05:00 oracleofnj: Clean up discrepancies
690 a18d5fc 2016-11-08T11:38:22-05:00 oracleofnj: Fix bug with x[-1]

```



691 962f812 2016-11-07T23:27:08-05:00 oracleofnj: Refactor scope for interpreter; resolve  
variables on demand; make selections work properly  
692 47bbef1 2016-11-06T22:05:55-05:00 oracleofnj: Minor adjustments to interpreter to work  
with mapped AST  
693 fddc6bc 2016-11-06T18:32:17-05:00 oracleofnj: Eliminate extraneous nulls in JSON  
694 ffddb17 2016-11-06T18:15:40-05:00 oracleofnj: Turn statement and function lists into  
StringMaps  
695 6810003 2016-11-05T19:47:57-04:00 oracleofnj: Fix pattern matching warning  
696 7107a46 2016-11-05T18:01:34-04:00 oracleofnj: Add function to check range literals for  
legality at parse time  
697 80b13d1 2016-11-05T15:13:10-04:00 oracleofnj: Handle selections better  
698 6cbb009 2016-11-04T15:48:58-04:00 oracleofnj: Count to 1,000,000 using tail-recursive  
versions of List.map and cartesian product  
699 9b2252d 2016-11-04T15:25:13-04:00 oracleofnj: Show enter and exit  
700 3585e43 2016-11-04T02:21:38-04:00 oracleofnj: See how high it can count recursively  
701 38cf541 2016-11-04T02:15:50-04:00 oracleofnj: Get the easy parts of the interpreter  
working  
702 5d81d6e 2016-11-03T17:17:51-04:00 oracleofnj: Start working on interpreter  
703 0078cee 2016-11-01T23:40:57-04:00 oracleofnj: Got a non-tail-recursive version of  
topological sort working  
704 85df175 2016-11-01T15:39:10-04:00 oracleofnj: Irrelevant highlighting thing  
705 84c719a 2016-11-01T14:39:49-04:00 oracleofnj: Rearrange nested functions  
706 557dc4e 2016-11-01T13:50:52-04:00 oracleofnj: Add circular import test case  
707 c476798 2016-11-01T13:35:46-04:00 oracleofnj: Fix syntax errors  
708 af5a31d 2016-11-01T13:31:49-04:00 GitHub: Merge pull request #37 from Neitsch/import-  
rec  
709 d451cc4 2016-11-01T13:31:33-04:00 GitHub: Merge pull request #38 from Neitsch/import-  
load  
710 02ca24f 2016-11-01T13:30:47-04:00 GitHub: Merge pull request #39 from Neitsch/wild-exc  
711 6fa0e39 2016-10-31T16:43:17-04:00 Neitsch: Raising exceptions on certain values  
712 e673dca 2016-10-31T15:56:43-04:00 Neitsch: Loading data from all imports  
713 6a28c05 2016-10-31T15:40:41-04:00 Neitsch: Recursively looking up dependencies  
714 3f28289 2016-10-31T11:53:10-04:00 GitHub: Merge pull request #36 from Neitsch/import-  
arrange  
715 4eae3b 2016-10-31T11:01:00-04:00 Neitsch: Removed obsolete parts  
716 7d7ble5 2016-10-31T10:59:12-04:00 Neitsch: Added unsorted function, globals and  
imports  
717 7d70af2 2016-10-30T15:23:04-04:00 oracleofnj: Add some explanatory comments  
718 40d6b16 2016-10-30T15:03:32-04:00 oracleofnj: More expansion samples  
719 af9b01c 2016-10-30T14:48:44-04:00 oracleofnj: Refactor expansion code  
720 903bc3f 2016-10-30T00:19:10-04:00 oracleofnj: Add test output  
721 68b7b03 2016-10-30T00:17:02-04:00 oracleofnj: Add test case  
722 a8bdf33 2016-10-30T00:04:05-04:00 oracleofnj: Add LHS slice expansion  
723 4ee6fdf 2016-10-29T17:36:17-04:00 oracleofnj: Add output  
724 2b8bcd 2016-10-29T17:27:22-04:00 oracleofnj: Expand dimension expressions  
725 443a818 2016-10-26T16:31:51-04:00 GitHub: Merge pull request #35 from ishaankolluri/  
master  
726 9ba3c65 2016-10-26T16:31:00-04:00 Ishaan Kolluri: Add UNIs  
727 022e8cd 2016-10-26T16:25:57-04:00 GitHub: Merge pull request #34 from ishaankolluri/  
master  
728 808aae5 2016-10-26T16:22:10-04:00 Ishaan Kolluri: Added change to precedence operators  
729 0bd9c4a 2016-10-26T15:59:53-04:00 GitHub: Merge pull request #33 from Neitsch/final-  
slicing-comments  
730 fb2b382 2016-10-26T15:54:11-04:00 oracleofnj: Thats all for now folks  
731 e7020ec 2016-10-26T15:00:11-04:00 GitHub: Merge pull request #32 from Neitsch/final-  
lrn-edits

732 4683f14 2016-10-26T14:48:41-04:00 oracleofnj: Flesh out switch expressions, add  
precedence  
733 4b7984a 2016-10-26T11:15:03-04:00 GitHub: Merge pull request #31 from Neitsch/more-lrm  
-edits  
734 3d587c5 2016-10-26T11:10:15-04:00 oracleofnj: Incorporate requested edits and a few  
more clarifications  
735 0c42b9c 2016-10-26T09:22:08-04:00 GitHub: Merge pull request #30 from ishaankolluri/  
LRM\_update  
736 cd81040 2016-10-26T03:30:20-04:00 ishaankolluri: Added changes to first half of LRM  
737 63fb02b 2016-10-26T02:13:17-04:00 GitHub: Merge pull request #29 from Neitsch/lrm-  
edits  
738 0941e96 2016-10-26T02:04:47-04:00 oracleofnj: Rebuild PDF  
739 cb04069 2016-10-26T02:04:01-04:00 oracleofnj: Add built in functions  
740 4abf638 2016-10-26T01:56:38-04:00 oracleofnj: Add built in functions  
741 7661925 2016-10-26T00:04:22-04:00 oracleofnj: Initial comments  
742 5932551 2016-10-25T21:30:40-04:00 GitHub: Merge pull request #28 from Neitsch/func-doc  
-fix  
743 cc66297 2016-10-25T20:14:27-04:00 Nigel Schuster: Fixed mistakes in functions part of  
the doc  
744 b978f00 2016-10-25T13:04:05-04:00 GitHub: Merge pull request #27 from ishaankolluri/  
master  
745 125a5bb 2016-10-25T12:49:38-04:00 Ishaan Kolluri: Removed AUX file  
746 2e1ea60 2016-10-25T11:30:35-04:00 GitHub: Merge pull request #26 from Neitsch/better-  
regex  
747 84b03ee 2016-10-25T01:22:31-04:00 oracleofnj: Fix let order  
748 91b40c5 2016-10-25T01:14:43-04:00 oracleofnj: Improve regex  
749 eb24036 2016-10-24T23:55:38-04:00 GitHub: Merge pull request #23 from Neitsch/file-io  
750 991c918 2016-10-24T23:20:12-04:00 oracleofnj: Replace fopen, fclose etc. with open,  
close etc.  
751 338faa0 2016-10-24T23:14:30-04:00 oracleofnj: Fix file inclusion and rebuild PDF  
752 b24edd3 2016-10-24T23:11:50-04:00 oracleofnj: Merge in expressions section  
753 44alcc5 2016-10-24T23:06:07-04:00 oracleofnj: Merge scanner changes and add regex to  
properly escape strings  
754 2f09a64 2016-10-24T15:52:10-04:00 Kevin: Added the Expression Section 4 to LRM  
755 1ea3c28 2016-10-24T15:26:16-04:00 oracleofnj: Merge branch 'master' into file-io  
756 ec7cc9c 2016-10-24T15:21:23-04:00 Jared Samet: Replace repetitive code with more  
idiomatic OCaml  
757 8cd39ac 2016-10-24T11:05:33-04:00 Kevin: Added string literals to scanner  
758 e5d2478 2016-10-24T11:00:39-04:00 Kevin: Added string literals to scanner  
759 a692466 2016-10-24T01:09:21-04:00 oracleofnj: Fix tests until strings ready  
760 8553a50 2016-10-24T01:08:29-04:00 oracleofnj: Fix tests until string ready  
761 0ed4ad7 2016-10-24T00:55:08-04:00 oracleofnj: Add File IO, Entry point and Example to  
LRM  
762 71e0b1c 2016-10-23T22:58:21-04:00 oracleofnj: Fix section reference  
763 92ac506 2016-10-23T22:39:06-04:00 Ishaan Kolluri: Make small change to data type  
section  
764 6abb290 2016-10-23T22:34:42-04:00 oracleofnj: Initial commit for File I/O section  
765 67b4b65 2016-10-23T19:30:03-04:00 Nigel Schuster: Reduce eye pain  
766 2824ee9 2016-10-23T19:03:24-04:00 GitHub: Merge pull request #20 from Neitsch/samples  
767 f8ae543 2016-10-23T18:23:11-04:00 GitHub: Merge branch 'master' into samples  
768 13d0896 2016-10-23T18:20:03-04:00 GitHub: Merge pull request #19 from Neitsch/sequence  
-operator  
769 e0c702d 2016-10-23T18:17:58-04:00 Neitsch: Fixed .gitignore  
770 3a2cd60 2016-10-23T18:16:35-04:00 GitHub: Merge branch 'master' into sequence-operator  
771 e42fe94 2016-10-23T18:05:48-04:00 Neitsch: Added code in LRM to test code samples  
772 9d2cd17 2016-10-23T17:24:15-04:00 Neitsch: Merge branch 'master' into samples

```

773 167ddd2 2016-10-23T17:18:35-04:00 Neitsch: Removed test output
774 57319c4 2016-10-23T17:11:13-04:00 oracleofnj: Remove intermediate files
775 53824ea 2016-10-23T17:10:39-04:00 oracleofnj: Flip precedence of -> and ?: (?: is now
lowest)
776 7dedf93 2016-10-23T17:05:23-04:00 oracleofnj: Add sequence operator to scanner/parser/
AST
777 9805753 2016-10-23T17:01:31-04:00 GitHub: Merge pull request #17 from Neitsch/make-
correction
778 e0c7aed 2016-10-23T16:59:33-04:00 Neitsch: Fixed test
779 ec3d682 2016-10-23T16:41:00-04:00 GitHub: Merge branch 'master' into make-correction
780 ea05658 2016-10-23T16:40:24-04:00 Neitsch: Moved sequence file
781 0ca56a0 2016-10-23T16:10:14-04:00 Neitsch: Merge
782 9d1094e 2016-10-23T16:08:59-04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
config
783 0a28413 2016-10-23T16:08:59-04:00 Neitsch: Completed initial functions section doc
784 0797f32 2016-10-23T16:08:12-04:00 Neitsch: Changed subsection header
785 9df31f7 2016-10-23T16:08:12-04:00 Neitsch: Added dimension section
786 8939903 2016-10-23T16:07:26-04:00 Neitsch: Started working on Functions
787 cae3b37 2016-10-23T16:06:27-04:00 Neitsch: Added dimension section
788 049c95d 2016-10-23T16:06:08-04:00 Neitsch: Started working on Functions
789 84d20b5 2016-10-23T16:01:00-04:00 Neitsch: Comparing sample code with correctly parsed
code in samples_comp
790 3f015ee 2016-10-23T15:52:01-04:00 GitHub: Merge pull request #18 from Neitsch/grammar-
bug-fixes
791 7e558c1 2016-10-23T15:44:20-04:00 GitHub: Merge branch 'master' into make-correction
792 edf3dea 2016-10-23T15:44:20-04:00 GitHub: Merge branch 'master' into grammar-bug-fixes
793 d4961eb 2016-10-23T15:43:16-04:00 GitHub: Merge pull request #15 from Neitsch/
functions-doc
794 0e0bda5 2016-10-23T15:05:42-04:00 GitHub: Merge branch 'master' into functions-doc
795 4652c67 2016-10-23T15:00:35-04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
config
796 b45718d 2016-10-23T02:27:36-04:00 oracleofnj: Modify grammar to allow [m,n] foo, bar,
baz;
797 143fcba 2016-10-22T23:23:10-04:00 GitHub: Merge pull request #16 from Neitsch/more-AST
798 a726236 2016-10-22T20:51:27-04:00 oracleofnj: Add comments and sample program
799 8db4098 2016-10-22T19:44:48-04:00 oracleofnj: Fix minor grammar bug
800 80754c3 2016-10-22T18:19:27-04:00 oracleofnj: Hook up scanner and parser
801 660de8c 2016-10-22T13:54:32-04:00 GitHub: Add stuff to the grammar, minor corrections
(#14)
802 cfe827d 2016-10-21T20:50:51-04:00 Nigel Schuster: Completed initial functions section
doc
803 3609366 2016-10-20T21:14:00-04:00 GitHub: Update scanner.mll
804 0d57652 2016-10-20T21:10:27-04:00 Kevin: Fixed bug in scanner
805 1848813 2016-10-20T20:21:49-04:00 Kevin: Made scanner
806 1b610ac 2016-10-20T13:50:22-04:00 Nigel Schuster: Merge
807 acb9b93 2016-10-20T13:44:06-04:00 Nigel Schuster: Changed subsection header
808 b95d039 2016-10-20T13:43:51-04:00 Nigel Schuster: Added dimension section
809 71b93bb 2016-10-20T13:43:09-04:00 Nigel Schuster: Started working on Functions
810 a15772c 2016-10-20T13:38:08-04:00 GitHub: Merge pull request #10 from ishaankolluri/
LRM
811 dee63c7 2016-10-20T13:26:28-04:00 GitHub: Merge pull request #1 from Neitsch/grammar-
doc
812 dc93dbf 2016-10-20T13:18:29-04:00 Nigel Schuster: Grammar import
813 4d763cb 2016-10-20T12:44:52-04:00 Ishaan Kolluri: Made refactor and edits to intro
section of LRM
814 e7443cc 2016-10-20T11:46:54-04:00 Ishaan Kolluri: Merging

```

815 7542b5d 2016-10-20T11:16:35-04:00 Nigel Schuster: Added dimension section  
816 995cf83 2016-10-19T12:28:09-04:00 Nigel Schuster: Started working on Functions  
817 40c2a5a 2016-10-19T03:43:06-04:00 ishaankolluri: Initial LRM Commit part 1  
818 02a5c17 2016-10-18T18:38:21-04:00 Ishaan Kolluri: Added LRM initial info  
819 d8794e9 2016-10-17T19:47:42-04:00 GitHub: Merge pull request #9 from Neitsch/  
documentation  
820 70aalb9 2016-10-16T13:36:23-04:00 Nigel Schuster: Added PDF Latex template  
821 5111202 2016-10-14T19:59:45-04:00 GitHub: Added a bunch of stuff to the grammar: (#8)  
822 da967e4 2016-10-12T13:24:50-04:00 Jared Samet: CFG Grammar (#6)  
823 fea4e4b 2016-10-08T11:42:39-04:00 GitHub: There is no need to constantly build all  
branches. (#2)  
824 7a5ccfc 2016-10-08T11:31:31-04:00 Nigel Schuster: Added greeting and newlines (#4)  
825 10b17f7 2016-10-08T11:31:08-04:00 GitHub: Imported microc (#5)  
826 726456f 2016-09-20T09:45:07-04:00 Nigel Schuster: [test] Add sample greeting to repo  
(#3)  
827 9a2183d 2016-09-15T18:44:00-04:00 Nigel Schuster: Added merlin config  
828 163e176 2016-09-14T18:51:53-04:00 Nigel Schuster: Moved whole build to script  
829 d401eea 2016-09-14T18:43:58-04:00 Nigel Schuster: Added oasis opam package  
830 ba7fd9c 2016-09-14T18:38:58-04:00 Nigel Schuster: Added ocaml configure (maybe this  
helps travis)  
831 a461eae 2016-09-14T18:26:10-04:00 Nigel Schuster: Configuring opam environment for  
travis  
832 ba2df2f 2016-09-14T18:19:26-04:00 Nigel Schuster: Added ocaml native compiler to apt  
package list  
833 a8e5958 2016-09-14T17:24:36-04:00 Nigel Schuster: Added some more (possibly necessary  
opam packages  
834 c54f5e3 2016-09-14T17:18:32-04:00 Nigel Schuster: Missed opam option  
835 b10adf0 2016-09-14T17:13:57-04:00 Nigel Schuster: Fixed opam install  
836 124f7f3 2016-09-14T17:08:09-04:00 Nigel Schuster: Fixed YML error  
837 4909fa8 2016-09-14T17:03:54-04:00 Nigel Schuster: Using avsm source  
838 4b24046 2016-09-14T16:58:33-04:00 Nigel Schuster: Allow sudo  
839 e7b50db 2016-09-14T16:56:57-04:00 Nigel Schuster: Fixed setup order  
840 f6d7ac4 2016-09-14T16:50:02-04:00 Nigel Schuster: Manually installing apt packages  
841 f4084ab 2016-09-14T16:40:55-04:00 Nigel Schuster: Test commit  
842 d7c5e9a 2016-09-14T13:15:43-04:00 Nigel Schuster: Initial commit

## 12. Special Thanks

We'd like to thank Bruce Verderaime for the [gdchart](#) library, which we modified and shipped to provide Extend with graph plotting functionality. Additionally, we'd like to credit Thomas Boutell for the `gd` library, on which `gdchart` relies. The copyright notice is in the repository.