Extend Language Final Report

Ishaan Kolluri	Jared Samet	Nigel Schuster	Kevin Ye
isk2108	jss2272	ns3158	ky2294
Project Manager	Language Guru	System Architect	Tester

December 20, 2016

Contents

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, theremby 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 nth cell of profit is calculated by subtracting the nth cells of cost from the nth 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: **, *, /, %, +, -

12

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(conditional, 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!/D11
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 multiples 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 dynamically-typed, statically-scoped, declarative language that brings the semantics of an interactive spreadsheet application to a compiled language. Extend features immutable values and allows the developer to write code without explicitly specifying the order of computations. It goes beyond typical spreadsheet applications by allowing the developer to encapsulate related sets of computations in reusable functions. 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-
          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  }
```

Identifiers

A function or variable name must begin with a lowercase or uppercase letter and can be followed by any number of letters, digits, or underscores.

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 is a subset of the cells of a variable, as described in section 3.5. Ranges can be nested arbitrarily deeply and can be used to represent (immutable) lists, matrices, or more complicated data structures. For convenience, the range literal syntax can be used to implicitly declare an anonymous variable and assign the range to the entire contents of this variable.

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
~	Bitwise NOT	Performs a bitwise negation on the binary representation of an expression.
-	Unary negation	A simple negative sign to negate expressions.
**	Power	Returns the first expression raised to the power of the second expression
*	Multiplication	Multiplies two expressions
/	Division	Divides first expression by second.
%	Modulo	Finds the remainder by dividing the expression on the left side of the modulo by the right side expression.
«	Left Shift	Performs a bitwise left shift on the binary representation of an expression.
»	Right Shift	Performs a sign-propagating bitwise right shift on the binary representation of an expression.
&	Bitwise AND	Performs a bitwise AND between two expressions.
+	Addition	Adds two expressions together.
-	Subtraction	Subtracts second expression from first.
|	Bitwise OR	Performs a bitwise OR between two expressions.
^	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. !empty evaluates to empty. It has equal precedence with 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. empty == empty evaluates to 1. Strings are compared by value.
!=	Not equals	x != y is equivalent to !(x == y).
<	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 empty.
>	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 empty, evaluates to 0 or empty respectively. Otherwise, if the second expression is truthy, falsey, or empty, evaluates to 1, 0, or empty respectively.
||	Short-circuit Logical OR	If the first expression is truthy or empty, evaluates to 1 or empty respectively. Otherwise, if the second expression is truthy, falsey, or empty, evaluates to 1, 0, or empty 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-expr`s is truthy, or `default-expr` if none of the `case-expr`s apply. If `switch-expr` is present, the `switch` expression evaluates to the `match-expr` for the first case where one of the `case-expr`s is equal (with equality defined as for the == operator) to `switch-expr`, or `default-expr` if none of the `case-expr`s 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
`size(expr)`	`Dimensions`	Evaluates to a Range consisting of one row and two columns; the first cell contains the number of rows of `expr` and the second contains the number of columns. If `expr` is a Number, a String, or Empty, both cells will contain 1.
`typeof(expr)`	`Value Type`	Evaluates to "Number", "String", "Range", or "Empty".
`row()`	`Row Location`	No arguments; returns the row of the cell that is being calculated
`column()`	`Column Location`	No arguments; returns the column of the cell that is being calculated
`+`	`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. Extend is an applicative language: 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 some or all of the cells of a variable or another 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 :=
3       {"Alpha", "Bravo", "Charlie", "Delta", "Echo";
4        "Foxtrot", "Golf", "Hotel", "India", "Juliett";
5        "Kilo", "Lima", "Mike", "November", "Oscar";
6        "Papa", "Quebec", "Romeo", "Sierra", "Tango"};
7
8     [3,3] bar;
9     bar[0,0] = foo[0,2]; // "Charlie"
10    bar[0,1] = foo[0,:]; // {"Alpha", "Bravo", "Charlie", "Delta", "Echo"}
11    bar[0,2] = foo[:,2]; // {"Charlie"; "Hotel"; "Mike"; "Romeo"}
12    bar[1,1] = foo[[1],[2]]; // "November" — the [1] indicates relative
13                             // In this case, works out to foo[2,3]
14
15    bar[1,2] = foo[3,];    // "Romeo" since foo has multiple columns
16    bar[2,2] = foo[2:[2],[−1]]; // {"Lima"; "Quebec"}
17
18    /* In this example, each cell of spam would be equal to the cell
19     * in ham in the equivalent location plus 1. */
20    ham := {2,4,6; 10,11,12; 20,30,40};
21    [3,3] spam := #ham + 1; // {3,5,7; 11,12,13; 21,31,41}
22
23    /* In this example, more_cookies would be a 3x4 range where in each row,
24     * the value is equal to the value in cookies in the same column.
25     * In other words, each row of more_cookies would be a copy of cookies. */
26    cookies := {"Chocolate","Oatmeal","Vanilla","Peanut Butter"};
27    [3,4] more_cookies := #cookies;
28
29    /* In this example, the values of baz would be
30     * 11, 12, 13 in the first row;
31     * 21, 22, 23 in the second row;
32     * 31, 32, 33 in the third row. */
33    ones := {1,2,3}; // 1 row, 3 columns
34    tens := {10;20;30}; // 3 rows, 1 column
35    [3,3] nums := #ones + #tens; // Equivalent to ones[0,[0]] + tens[[0],0]
36
37    return 0;
```

```
38   }
```

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 immutable and lazily evaluated cell values, 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 set 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 variables, 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 variable with 2 rows and 5 columns
2   [m, n] bar; // Declares bar as a variable with m rows and n columns
3   [3, 3] ham, eggs, spam; // Declares ham, eggs and spam as distinct 3x3 variables
4   baz; // Declares baz as a variable with 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 variable,

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 variables
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 variables; 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 Cell 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. A cell's value is needed when a range expression consists of that single cell, or when the cell belongs to a range that is assigned as the value for another cell. In other words, an intermediate range expression that consists of multiple cells will not cause the constituent cells to be evaluated; however, a range expression that has one row and one column will cause that one cell's value to be evaluated. In conditional expressions and in short-circuiting operator expressions, only the predicate and the relevant conditional branch will be evaluated. In an expression using the precedence operator, the preceding expression will be evaluated before the succeeding expression. 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

File I/O

```
1  open(filename, mode) — returns a file handle for use with the other file I/O functions
2  close(file_handle) — close a file handle
3  read(file_handle, num_bytes) — reads num_bytes from a file; 0 reads entire file
4  readline(file_handle) — read until the first newline
5  write(file_handle, buffer) — write the contents of buffer (a String) to the handle
6  STDIN, STDOUT, STDERR — global variables initialized to the appropriate file handles
7  print_endline(val) — convert val to a string and write to STDOUT
```

Math Functions - Imported straight from C

```
1  sin(x), cos(x), tan(x), acos(x), asin(x), atan(x), sinh(x), cosh(x), tanh(x),
2  exp(x), log(x), log10(x), sqrt(x), ceil(x), fabs(x), floor(x), isNaN(x)
3  random() — Just for fun — very non—random.
```

Math Functions - Not imported from C

```
1   isInfinite(x) — returns —1 for —infinity, 0 for finite, or 1 for +infinity
2   round(val, number_of_digits);
3   gcd(m, n) — returns the GCD of two numbers
4   lcm(m, n) — returns the LCM of two numbers
5   sign(arg) — returns —1, 0, or 1
6   sum(rng) — adds all the numbers in rng
7   nmax(n1, n2) — returns the max of two numbers
8   max(rng) — returns the largest number in a range
9   nmin(n1, n2) — returns the min of two numbers
10  min(rng) — returns the smallest number in a range
11  avg([m,n] rng) — return the average of the numbers in a range
12  stdev([m,n] rng) — return the standard deviation of the numbers in a range
13  sumsq(rng) — returns the sum of the squares of the numbers in rng
14  sumproduct([m,n] rng1, [m,n] rng2) — returns the inner product of rng1 and rng2
15  sumxmy2([m,n] rng1, [m,n] rng2) — returns the sum of squared differences between the
        elements of rng1 and rng2
16  mmult([m,n] rng1, [n,p] rng2) — multiplies two matrices
17  linest([p,q] known_ys, [p,q] known_xs) — performs a linear regression with known_ys as
         the dependent variables and known_xs as the independent variables
18  normalize([m,n] arg) — return the unit norm vector in the same direction as arg
```

String Functions

```
1   len(str) — returns the length of a String
2   toASCII(val) — returns a 1 x n range of the ASCII values of a String
3   fromASCII(val) — converts a 1 x n range of ASCII values into a String
4   parseFloat(str) — wrapper around C atof()
5   toUpper(text) — converts a string to uppercase
6   toLower(text) — converts a string to lowercase
7   left(str, num_chars) — returns the leftmost num_chars of str
8   right(str, num_chars) — returns the rightmost num_chars of str
9   substring(str, start, length) — returns a substring of str
10  repeat(str, num) — repeat a string, num times.
11  toString(arg) — convert any value into a String representation
12  ltrim(s) — remove whitespace at the beginning of s
13  rtrim(s) — remove whitespace at the end of s
14  trim(s) — remove whitespace on both ends of s
15  reverse(s) — reverses a string
16  padLeft(str, pad_char, total_length) — for a string shorter than total_length, pad on
        the left with pad_char
17  charAt(str, i) — return the ASCII code of the ith character of str
18  parseString(s) — best efforts to convert a string into the correct value
```

Plotting

```
1  bar_chart(file_handle, labels, vals);
2  line_chart(file_handle, labels, x_vals);
```

Range Functions

```
 1  transpose([m,n] rng) — transpose a matrix; works with any dimensions
 2  flatten([m,n] rng) — turn a rectangular range into a long row vector
 3  isNumber(x) — equal to typeof(x) == "Number"
 4  isEmpty(x) — equal to typeof(x) == "Number"
 5  colRange(start, end) — return a column vector with the integers from start to (end—1)
 6  rowRange(start, end) — return a row vector with the integers from start to (end—1)
 7  match(list, val) — finds the first occurence of val in list; list can be either a row
       or a column vector and does not need to be sorted
 8  bsearch(list, val) — finds the first occurrence of val in list; list must be a sorted
       column vector
 9  join([m,n] cells, joiner) — concatenate the string representation of either a column
       or a row vector, using joiner as the delimiter
10  joinRange([m,n] cells, rowJoiner, colJoiner) — concatenate a range, joining rows with
       rowJoiner and columns with colJoiner
11  numRows(arg) — return the number of rows in arg
12  numCols(arg) — return the number of columns in arg
13  split(string, splitter) — returns a row vector of strings using splitter (which must
       be a one—character String) as a delimiter
14  splitToRange(string, row_splitter, col_splitter) — returns a range of strings using
       row_splitter as the row delimiter and col_splitter as the column delimiter
15      case charAt(trimmed,0) == toASCII("{") && charAt(trimmed,—1) == toASCII("}"):
16  append([m,n] rg1, [p,q] rg2) — concatenate two ranges, horizontally
17  stack(rg1, rg2) — concatenate two ranges, vertically
18  mergesort([m,n] rng, sort_col) — return a sorted copy of rng, using sort_col for
       comparisons
```

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

Sep 11, 2016 – Dec 19, 2016

Contributions to master, excluding merge commits

Contributions: Commits
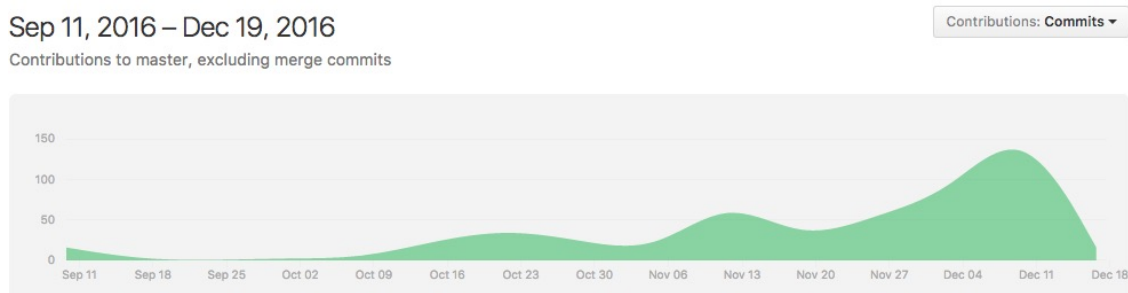
Github & Travis CI

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 indermediate 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.

An interesting tidbit is that the correct way to think about the distinction between variables and ranges did not become completely clear to us until quite late in the process. It only became clear when we finally coded up all the structure definitions in C and understood how they interacted.

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	oracleofnj
Nigel Schuster	development protocol, code generation, scripting	Neitsch
Ishaan Kolluri	initial LRM, Final Report, regression tests, stdlib functions, scripting	ishaankolluri
Kevin Ye	initial scanner, regression tests, stdlib functions	kevinye1

5. Extend's Internal Architecture
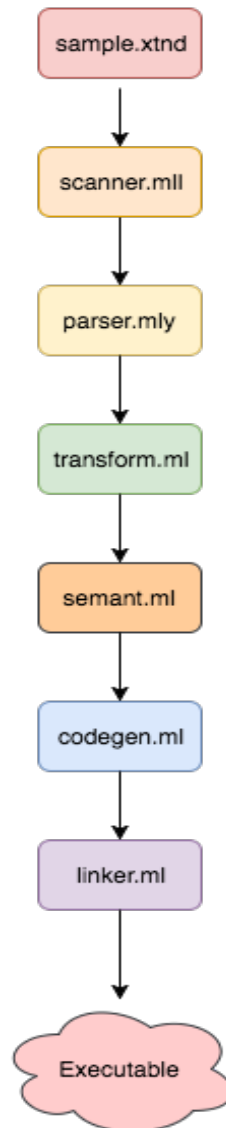
5.1 The Extend Compiler

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`: OCamlyacc 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; instead, we attempt to degrade gracefully by having expressions with incompatible types evaluate to empty. 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      // imagine foo is the 16th variable defined in the program
6      scope->defns = global_definitions[15];
7
8      // Create an array of pointers to variable instances; one
9      // pointer per variable. Only one variable in this function
10     scope->insts = new var_instance* [1];
11
12     // getVar calls instantiate_var if that instance pointer is still NULL,
13     // or just returns the pointer if it's already been instantiated.
14     // The instantiated variable keeps a copy of the pointer to its scope.
15     // The 2nd argument to getVar is the variable's index within the function.
16     var_instance *return_variable = getVar(scope, 0);
17
18     // Get the value of cell [0,0] of return_variable
19     return getVal(return_variable, 0, 0);
20   }
```

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 Travic 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      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  }
```

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
              [0,0] + " with a TSP of " + toString(player[0,1]) + "!") —>
19      bar_chart(open("barchart.png","wb"), transpose(players[:,0]), transpose(players
              [:,1]));
20  }
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],:] :
              tsp_ranking[[−1],0];
31    return tsp_ranking[m−1,0];
32  }
```

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 }
```

48

```
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  /* Ocamlyacc 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
          glob, List.rev fnc, List.rev ext) }
40
41 program_piece:
42     /* nothing */ {([],[],[],[])}
43   | program_piece import      { let (imp, glob, fnc, ext) = $1 in ($2 :: imp, glob,
        fnc, ext) }
```

50

```
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(SizeOf, $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
            $6, snd $6) }
163    | SWITCH LBRACE default_case_list RBRACE { Switch(None, fst $3, snd $3) }
164
165  switch_cond:
166      /* nothing */ { None }
167    | expr { Some $1 }
168
169  default_case_list:
170      case_list {(List.rev $1, Empty)}
171    | case_list default_expr {(List.rev $1, $2)}
172
173  case_list:
174      case_stmt { [$1] }
175    | case_list case_stmt { $2 :: $1 }
176
177  case_stmt:
178      CASE case_expr_list COLON expr SEMI { (List.rev $2, $4) }
179
180  default_expr:
181      DEFAULT COLON expr SEMI { $3 }
182
183  case_expr_list:
184      expr { [$1] }
185    | case_expr_list COMMA expr { $3 :: $1 }
186
187  func_expr:
188      ID LPAREN opt_arg_list RPAREN { Call($1, $3) }
189
190  range_expr:
191      LBRACE row_list RBRACE { allow_range_literal (LitRange(List.rev $2)) }
192
193  row_list:
194      col_list {[List.rev $1]}
195    | row_list SEMI col_list {List.rev $3 :: $1}
196
197  col_list:
198      expr {[$1]}
199    | col_list COMMA expr {$3 :: $1}
200
201  opt_arg_list:
202      /* nothing */ {[]}
203    | arg_list { List.rev $1 }
204
205  arg_list:
206      expr {[$1]}
207    | 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 ->
                                          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    | 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 (el, e) =
156      "{\"Cases\": " ^ string_of_list (Exprs el) ^ ", " ^
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
          \": " ^ string_of_index end_idx ^ "}"
165
166 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 ^ ", " ^
177                             "\"VarName\": " ^ quote_string s ^ "}"
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 ^ ", " ^
197                               "\"expr\": " ^ string_of_expr e ^ "}"
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 ^ "}}"
272
273  let allow_range_literal = function
274      LitRange(rowlist) ->
275        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_externs) =
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_externs) (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
              the new variable *)
63      Empty     -> raise (IllegalExpression("Empty not allowed in " ^ expr_loc))
64      | LitString(s) -> raise (IllegalExpression("String literal " ^ quote_string s ^ "
           not allowed in " ^ expr_loc))
65      | LitRange(rl) -> raise (IllegalExpression("Range literal " ^ string_of_list (Rows
             rl) ^ " not allowed in " ^ expr_loc))
66      | e           -> let new_id = idgen expr_loc in (
67          Id(new_id),
68          [Varinit (one_by_one, [(new_id, None)]);
69           Assign (new_id, zero_comma_zero, Some e)]) in
70
71    let expand_index index_loc = function
72      (* Expand one index of a slice if necessary. *)
73        Abs(e) -> let (new_e, new_stmts) = expand_expr index_loc e in
74        (Abs(new_e), new_stmts)
75      | DimensionStart -> (DimensionStart, [])
76      | DimensionEnd -> (DimensionEnd, [])
77      | Rel(_) -> raise (IllegalExpression("relative - this shouldn't be possible")) in
78
79    let expand_slice slice_loc = function
80      (* Expand one or both sides as necessary. *)
81        None -> (entire_dimension, [])
82      | Some (Some (Abs(e)), None) ->
83        let (start_e, start_stmts) = expand_expr (slice_loc ^ "_start") e in
84        ((Some (Abs(start_e)), None), start_stmts)
85      | Some (Some idx_start, Some idx_end) ->
86        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
            ") row_slice in
98        let (new_col_slice, col_exprs) = expand_slice (asgn_loc ^ "_" ^ var_name ^ "_col
            ") col_slice in
99        Assign(var_name, (Some new_row_slice, Some new_col_slice), formula) :: (
            row_exprs @ col_exprs)
100     with IllegalExpression(s) ->
101       raise (IllegalExpression("Illegal expression (" ^ s ^ ") in " ^
102                                 string_of_assign (var_name, (row_slice, col_slice),
                                    formula))) in
103
104   let expand_init (r, c) (v, e) =
105     Varinit((Some r, Some c), [(v, None)]) ::
106     match e with
107       None -> []
108     | Some e -> [Assign (v, entire_range, Some e)] in
109
110   let expand_dimension dim_loc = function
111       None -> expand_expr dim_loc (LitInt(1))
112     | Some e -> expand_expr dim_loc e in
113
114   let expand_varinit fname ((row_dim, col_dim), inits) =
115     (* expand_varinit: Take a Varinit and return a list of more atomic
116        statements. Each dimension will be given a temporary ID, which
117        will be declared as [1,1] _tmpXXX; the formula for tmpXXX will be
118        set as a separate assignment; the original variable will be
119        declared as [_tmpXXX, _tmpYYY] var; and the formula assignment
120        will be applied to [:,:]. *)
121     try
122       let (row_e, row_stmts) = expand_dimension (fname ^ "_" ^ (String.concat "_" (
            List.map fst inits)) ^ "_row_dim") row_dim in
123       let (col_e, col_stmts) = expand_dimension (fname ^ "_" ^ (String.concat "_" (
            List.map fst inits)) ^ "_col_dim") col_dim in
124       row_stmts @ col_stmts @ List.concat (List.map (expand_init (row_e, col_e)) inits
            )
125     with IllegalExpression(s) ->
126       raise (IllegalExpression("Illegal expression (" ^ s ^ ") in " ^
127                                 string_of_varinit ((row_dim, col_dim), inits))) in
128
129   let expand_stmt fname = function
130     Assign(a) -> expand_assign fname a
131   | Varinit(d, inits) -> expand_varinit fname (d, inits) in
132
133   let expand_stmt_list fname stmts = List.concat (List.map (expand_stmt fname) stmts)
        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
              )), Some e) ->
205         if StringMap.mem var_name m
206         then (let v = StringMap.find var_name m in
207               StringMap.add var_name {v with var_formulas = v.var_formulas @ [{
208                   formula_row_start = row_start;
209                   formula_row_end = row_end;
210                   formula_col_start = col_start;
211                   formula_col_end = col_end;
212                   formula_expr = e;
213               }]} m)
214         else raise (UnknownVariable(string_of_stmt (Assign(var_name, (Some (Some
                row_start, row_end), Some (Some col_start, col_end)), Some e))))
215       | Assign(a) -> raise (LogicError("Unrecognized format for post─desugaring Assign:
            " ^ string_of_stmt (Assign(a)))) in
216
217     let vds_of_stmts stmts =
218       let is_varinit = function Varinit(_,_) -> true | _ -> false in
219       let varinits = List.filter is_varinit stmts in
220       let vars_just_the_names = map_of_list (List.map vd_of_vi varinits) in
221       List.fold_left add_formula vars_just_the_names stmts in
222
223     let fd_of_raw_func f = (f.name, {
224         func_params = f.params;
225         func_body = vds_of_stmts f.body;
226         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 =
                lib_name})) lib_fns in
232
233    (vds_of_stmts globals,
234     map_of_list (List.map fd_of_raw_func functions),
235     map_of_list (List.concat (List.map tupleize_library externs)))
236
237  let single_formula e = {
238    formula_row_start = DimensionStart;
239    formula_row_end = Some DimensionEnd;
240    formula_col_start = DimensionStart;
241    formula_col_end = Some DimensionEnd;
242    formula_expr = e;
243  }
244
245  let ternarize_exprs (globals, functions, externs) =
246    let rec ternarize_expr lhs_var = function
247        BinOp(e1, LogAnd, e2) ->
248        let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
249        let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
250        (Ternary(UnOp(Truthy,new_e1), UnOp(Truthy,new_e2), LitInt(0)), new_e1_vars @
                new_e2_vars)
251      | BinOp(e1, LogOr, e2) ->
252        let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
253        let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
254        (Ternary(UnOp(Truthy,new_e1), LitInt(1), UnOp(Truthy,new_e2)), new_e1_vars @
                new_e2_vars)
255      | BinOp(e1, op, e2) ->
256        let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
257        let (new_e2, new_e2_vars) = ternarize_expr lhs_var  e2 in
258        (BinOp(new_e1, op, new_e2), new_e1_vars @ new_e2_vars)
259      | UnOp(op, e) ->
260        let (new_e, new_e_vars) = ternarize_expr lhs_var e in
261        (UnOp(op, new_e), new_e_vars)
262      | Ternary(cond, e1, e2) ->
263        let (new_cond, new_cond_vars) = ternarize_expr lhs_var cond in
264        let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
265        let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
266        (Ternary(new_cond, new_e1, new_e2), new_cond_vars @ new_e1_vars @ new_e2_vars)
267      | Call(fname, args) ->
268        let new_args_and_vars = List.map (ternarize_expr lhs_var) args in
269        (Call(fname, (List.map fst new_args_and_vars)), List.concat (List.map snd
                new_args_and_vars))
270      | Selection(e, (sl1, sl2)) ->
271        let (new_e, new_e_vars) = ternarize_expr lhs_var e in
272        let (new_sl1, new_sl1_vars) = ternarize_slice lhs_var sl1 in
273        let (new_sl2, new_sl2_vars) = ternarize_slice lhs_var sl2 in
274        (Selection(new_e, (new_sl1, new_sl2)), new_e_vars @ new_sl1_vars @ new_sl2_vars)
275      | Precedence(e1, e2) ->
276        let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
277        let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
278        (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(
                  LitInt(0))),None)))),
292             (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,
             combine_everything more_cases) in
304      (combine_everything new_cases, new_cond_vars @ new_case_vars @ new_dflt_vars)
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,
                  unify_switch_cond_and_case_conds switch_cond case_conds)) in
318          combined_expr
319        | [] -> raise(LogicError("Empty case condition list")) in
320      ((unify_switch_cond_and_case_conds cond new_conds, new_e),new_cond_vars @
           new_e_vars)
321    and ternarize_slice lhs_var = function
322        None -> (None, [])
323      | Some (i1, i2) ->
324        let (new_i1, new_i1_vars) = ternarize_index lhs_var i1 in
325        let (new_i2, new_i2_vars) = ternarize_index lhs_var i2 in
326        (Some (new_i1, new_i2), new_i1_vars @ new_i2_vars)
327    and ternarize_index lhs_var = function
328        Some Abs(e) ->
329        let (new_e, new_e_vars) = ternarize_expr lhs_var e in
330        (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.
                var_formulas in
340        ({vardef with var_formulas = List.map fst new_formulas_and_vars}, List.concat (
                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 ->
                ternarize_variable (fn_name ^ "_" ^ varname) vardef) m in
343        let add_item var_name (orig_var, new_vars) l = ((var_name, orig_var) :: fst l,
                new_vars :: snd l) in
344        let combined_list = StringMap.fold add_item new_variables_and_maps ([],[]) in
345        map_of_list (List.rev (fst combined_list) @ List.concat (snd combined_list)) in
346      let ternarize_function fn_name fn_def = {fn_def with func_body = ternarize_variables
                fn_name fn_def.func_body} in
347      (ternarize_variables "global" globals, StringMap.mapi ternarize_function functions,
            externs)
348
349    let reduce_ternaries (globals, functions, externs) =
350      let rec reduce_expr lhs_var = function
351        | BinOp(e1, op, e2) ->
352          let (new_e1, new_e1_vars) = reduce_expr lhs_var e1 in
353          let (new_e2, new_e2_vars) = reduce_expr lhs_var  e2 in
354          (BinOp(new_e1, op, new_e2), new_e1_vars @ new_e2_vars)
355        | UnOp(op, e) ->
356          let (new_e, new_e_vars) = reduce_expr lhs_var e in
357          (UnOp(op, new_e), new_e_vars)
358        | Ternary(cond, e1, e2) -> reduce_ternary lhs_var cond e1 e2
359        | Call(fname, args) ->
360          let new_args_and_vars = List.map (reduce_expr lhs_var) args in
361          (Call(fname, (List.map fst new_args_and_vars)), List.concat (List.map snd
                new_args_and_vars))
362        | Selection(e, (sl1, sl2)) ->
363          let (new_e, new_e_vars) = reduce_expr lhs_var e in
364          let (new_sl1, new_sl1_vars) = reduce_slice lhs_var sl1 in
365          let (new_sl2, new_sl2_vars) = reduce_slice lhs_var sl2 in
366          (Selection(new_e, (new_sl1, new_sl2)), new_e_vars @ new_sl1_vars @ new_sl2_vars)
367        | Precedence(e1, e2) ->
368          let (new_e1, new_e1_vars) = reduce_expr lhs_var e1 in
369          let (new_e2, new_e2_vars) = reduce_expr lhs_var e2 in
370          (Precedence(new_e1, new_e2), new_e1_vars @ new_e2_vars)
371        (* | Debug(e) ->
372          let (new_e, new_e_vars) = reduce_expr lhs_var e in
373          (Debug(new_e), new_e_vars) *)
374        | e -> (e, [])
375      and reduce_ternary lhs_var cond e1 e2 =
376        let (new_cond, new_cond_vars) = reduce_expr lhs_var cond in
377        let (new_true_e, new_true_vars) = reduce_expr lhs_var e1 in
378        let (new_false_e, new_false_vars) = reduce_expr lhs_var e2 in
379        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,
                 new_cond))]}) ::
385         (new_true_id, {lhs_vardef with var_formulas = [single_formula new_true_e]}) ::
386         (new_false_id, {lhs_vardef with var_formulas = [single_formula new_false_e]}) ::
387         (new_cond_vars @ new_true_vars @ new_false_vars))
388    and reduce_slice lhs_var = function
389        None -> (None, [])
390      | Some (i1, i2) ->
391        let (new_i1, new_i1_vars) = reduce_index lhs_var i1 in
392        let (new_i2, new_i2_vars) = reduce_index lhs_var i2 in
393        (Some (new_i1, new_i2), new_i1_vars @ new_i2_vars)
394    and reduce_index lhs_var = function
395        Some Abs(e) ->
396        let (new_e, new_e_vars) = reduce_expr lhs_var e in
397        (Some(Abs(new_e)), new_e_vars)
398      | Some Rel(e) ->
399        let (new_e, new_e_vars) = reduce_expr lhs_var e in
400        (Some(Rel(new_e)), new_e_vars)
401      | i -> (i, []) in
402    let reduce_formula lhs_var f =
403      let (new_expr, new_vars) = reduce_expr lhs_var f.formula_expr in
404      ({f with formula_expr = new_expr}, new_vars) in
405    let reduce_variable varname vardef =
406      let new_formulas_and_vars = List.map (reduce_formula (varname, vardef)) vardef.
             var_formulas in
407      ({vardef with var_formulas = List.map fst new_formulas_and_vars}, List.concat (
             List.map snd new_formulas_and_vars)) in
408    let reduce_variables fn_name m =
409      let new_variables_and_maps = StringMap.mapi (fun varname vardef -> reduce_variable
             (fn_name ^ "_" ^ varname) vardef) m in
410      let add_item var_name (orig_var, new_vars) l = ((var_name, orig_var) :: fst l,
             new_vars :: snd l) in
411      let combined_list = StringMap.fold add_item new_variables_and_maps ([],[]) in
412      map_of_list (List.rev (fst combined_list) @ List.concat (snd combined_list)) in
413    let reduce_function fn_name fn_def = {fn_def with func_body = reduce_variables
           fn_name fn_def.func_body} in
414    (reduce_variables "global" globals, StringMap.mapi reduce_function functions,
           externs)
415
416  let create_ast filename =
417    let ast_imp_res = expand_file true filename in
418    let ast_expanded = expand_expressions ast_imp_res in
419    let ast_mapped = create_maps ast_expanded in check_semantics ast_mapped ;
420    let ast_ternarized = ternarize_exprs ast_mapped in
421    let ast_reduced = reduce_ternaries ast_ternarized in check_semantics ast_reduced ;
422    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
            -> GlobalVariable(accum_idx) | ExtendFunctions -> ExtendFunction(accum_idx) in
31      (StringMap.add key index_val accum_map, accum_idx + 1) in
32    StringMap.fold add_item m (StringMap.empty, 0)
33
34  let create_symbol_table global_symbols fn_def =
35    let (local_indices, _) = index_map Locals fn_def.func_body in
36    let add_param (st, idx) param_name =
37      let new_st = StringMap.add param_name (FunctionParameter(idx)) st in
38      (new_st, idx + 1) in
39    let (params_and_globals, _) = List.fold_left add_param (global_symbols, 0) (List.map
          snd fn_def.func_params) in
40    StringMap.fold StringMap.add local_indices params_and_globals
41
42  let check_semantics (globals, functions, externs) =
43    let fn_signatures = map_of_list
44        ((StringMap.fold (fun s f l -> (s, List.length f.func_params) :: l) functions
            []) @
45         (StringMap.fold (fun s f l -> (s, List.length f.extern_fn_params) :: l) externs
            [])) in
46    let (global_symbols, _) = index_map Globals globals in
47
48    let check_call context called_fname num_args =
49      if (not (StringMap.mem called_fname fn_signatures)) then
50        (print_endline ("In " ^ context ^ "(), the undefined function " ^ called_fname ^
            "() was called") ;
51         raise(UnknownFunction(context ^ "," ^ called_fname)))
52      else let signature_args = StringMap.find called_fname fn_signatures in
```

69

```
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, (sl1, sl2)) -> check_expr fname symbols e ; check_slice fname
                symbols sl1 ; check_slice fname symbols sl2
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 : Llvm.lltype;*)
38    float_t : Llvm.lltype;
39    rhs_index_t : Llvm.lltype;
40    rhs_slice_t : Llvm.lltype;
41    rhs_selection_t : Llvm.lltype;
42    rhs_index_p : Llvm.lltype;
43    rhs_slice_p : Llvm.lltype;
44    rhs_selection_p : Llvm.lltype;
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
           D Matrix of values*)
152    and subrange_t = Llvm.named_struct_type ctx "subrange" (*Subrange is a wrapper
           around a range to cut cells*)
153    and int_t = Llvm.i32_type ctx (*Integer*)
154    and long_t = Llvm.i64_type ctx
155    and float_t = Llvm.double_type ctx
156    and flags_t = Llvm.i8_type ctx (*Flags for statuses*)
157    and char_t = Llvm.i8_type ctx (*Simple ASCII character*)
158    and bool_t = Llvm.i1_type ctx (*boolean 0 = false, 1 = true*)
159    and void_t = Llvm.void_type ctx (**)
160    and value_t = Llvm.named_struct_type ctx "value" (*Value encapsulates the content of
            a cell*)
161    and dimensions_t = Llvm.named_struct_type ctx "dimensions" (**)
162    and resolved_formula_t = Llvm.named_struct_type ctx "resolved_formula"
163    and extend_scope_t = Llvm.named_struct_type ctx "extend_scope"
164    and var_defn_t = Llvm.named_struct_type ctx "var_def"
165    and formula_t = Llvm.named_struct_type ctx "formula"
166    and string_t = Llvm.named_struct_type ctx "string" in
167    let var_instance_p = (Llvm.pointer_type var_instance_t)
168    and var_defn_p = Llvm.pointer_type var_defn_t
169    and resolved_formula_p = (Llvm.pointer_type resolved_formula_t)
170    and subrange_p = (Llvm.pointer_type subrange_t)
171    and value_p = (Llvm.pointer_type value_t)
172    and value_p_p = (Llvm.pointer_type (Llvm.pointer_type value_t))
173    and extend_scope_p = (Llvm.pointer_type extend_scope_t)
174    and char_p = (Llvm.pointer_type char_t)
175    and string_p = (Llvm.pointer_type string_t)
176    and char_p_p = (Llvm.pointer_type (Llvm.pointer_type char_t))
177    and string_p_p = (Llvm.pointer_type (Llvm.pointer_type string_t))
178    and number_t = float_t
179    and formula_p = (Llvm.pointer_type formula_t) in
180    let rhs_index_t = Llvm.named_struct_type ctx "rhs_index"
181    and rhs_slice_t = Llvm.named_struct_type ctx "rhs_slice"
182    and rhs_selection_t = Llvm.named_struct_type ctx "rhs_selection" in
183    let rhs_index_p = Llvm.pointer_type rhs_index_t
184    and rhs_slice_p = Llvm.pointer_type rhs_slice_t
185    and rhs_selection_p = Llvm.pointer_type rhs_selection_t
186    (*and void_p = (Llvm.pointer_type void_t)*) in
187    let var_instance_p_p = (Llvm.pointer_type var_instance_p)
188    and formula_call_t = (Llvm.function_type value_p [|extend_scope_p(*scope*); int_t(*
           row*); int_t(*col*)|]) in
189    let formula_call_p = Llvm.pointer_type formula_call_t in
190    let _ = Llvm.struct_set_body rhs_index_t (Array.of_list [
191        value_p (*val_of_expr*);
192        char_t (*rhs_index_type*);
193      ]) false in
194    let _ = Llvm.struct_set_body rhs_slice_t (Array.of_list [
195        rhs_index_p (*slice start index*);
196        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)
             the_module
31      in Hashtbl.add runtime_functions fname the_func in
32    add_runtime_func "strlen" bt.long_t [|bt.char_p|];
33    add_runtime_func "strcmp" bt.long_t [|bt.char_p; bt.char_p|];
34    add_runtime_func "pow" bt.float_t [|bt.float_t; bt.float_t|] ;
35    add_runtime_func "lrint" bt.int_t [|bt.float_t|] ;
36    add_runtime_func "llvm.memcpy.p0i8.p0i8.i64" bt.void_t [|bt.char_p; bt.char_p; bt.
        long_t; bt.int_t; bt.bool_t|] ;
37    add_runtime_func "incStack" bt.void_t [||] ;
38    add_runtime_func "getVal" bt.value_p [|bt.var_instance_p; bt.int_t; bt.int_t|] ;
39    add_runtime_func "rg_eq" bt.int_t [|bt.value_p; bt.value_p|] ;
40    add_runtime_func "clone_value" bt.value_p [|bt.value_p;|] ;
41    (* add_runtime_func "freeMe" (Llvm.void_type ctx) [|bt.extend_scope_p;|] ; *)
42    add_runtime_func "getSize" bt.value_p [|bt.var_instance_p;|] ;
43    add_runtime_func "get_variable" bt.var_instance_p [|bt.extend_scope_p; bt.int_t|] ;
44    add_runtime_func "null_init" (Llvm.void_type ctx) [|bt.extend_scope_p|] ;
45    add_runtime_func "debug_print" (Llvm.void_type ctx) [|bt.value_p ; bt.char_p|] ;
46    add_runtime_func "new_string" bt.value_p [|bt.char_p|] ;
47    add_runtime_func "deref_subrange_p" bt.value_p [|bt.subrange_p|];
48    add_runtime_func "debug_print_selection" (Llvm.void_type ctx) [|bt.rhs_selection_p
        |];
49    add_runtime_func "extract_selection" bt.value_p [|bt.value_p; bt.rhs_selection_p; bt
        .int_t; bt.int_t|];
50    add_runtime_func "box_command_line_args" bt.value_p [|bt.int_t; bt.char_p_p|];
51    add_runtime_func "verify_assert" (Llvm.void_type ctx) [|bt.value_p; bt.char_p|];
52    ()
53
```

```
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 (
             fun a -> base_types.value_p) func.extern_fn_params)) in
69      let llvm_fname = "extend_" ^ fname in
70      let llvm_fn = Llvm.declare_function llvm_fname llvm_ftype base_module in
71      StringMap.add fname llvm_fn accum_map in
72    let library_functions = StringMap.fold declare_library_function externs StringMap.
         empty in
73    let define_user_function fname func =
74      let llvm_fname = "extend_" ^ fname in
75      let llvm_ftype = Llvm.function_type base_types.value_p (Array.of_list (List.map (
             fun a -> base_types.value_p) func.func_params)) in
76      let llvm_fn = Llvm.define_function llvm_fname llvm_ftype base_module in
77      (func, llvm_fn) in
78    let extend_functions = StringMap.mapi define_user_function functions in
79    let function_llvalues = StringMap.fold StringMap.add (StringMap.map snd
         extend_functions) library_functions in
80
81    (* Build the global symbol table *)
82    let (global_symbols, num_globals) = index_map Globals globals in
83    let (extend_fn_numbers, num_extend_fns) = index_map ExtendFunctions extend_functions
          in
84
85    (* Create the global array that will hold each function's array of var_defns. *)
86    let vardefn_ptr = Llvm.const_pointer_null base_types.var_defn_p in
87    let vardefn_array = Array.make (StringMap.cardinal extend_functions) vardefn_ptr in
88    let array_of_vardefn_ptrs = Llvm.define_global "array_of_vardefn_ptrs" (Llvm.
         const_array base_types.var_defn_p vardefn_array) base_module in
89
90    (* Create the pointer to the global scope object *)
91    let global_scope_loc = Llvm.define_global "global_scope_loc" (Llvm.
         const_pointer_null base_types.extend_scope_p) base_module in
92
93    let main_def = Llvm.define_function "main" (Llvm.function_type base_types.int_t [|
         base_types.int_t; base_types.char_p_p|]) base_module in
94    let main_bod = Llvm.builder_at_end context (Llvm.entry_block main_def) in
95
96    let init_def = Llvm.define_function "initialize_vardefns" (Llvm.function_type (Llvm.
         void_type context) [|||]) base_module in
97    let init_bod = Llvm.builder_at_end context (Llvm.entry_block init_def) in
98
99    let literal_def = Llvm.define_function "initialize_literals" (Llvm.function_type (
         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 instanciates 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"
                  old_builder in
187          let _ = (llvm_var $> (subrange_ptr, (subrange_field_index BaseRangePtr)))
                  old_builder in
188          let _ = ((Llvm.const_null base_types.int_t) $> (subrange_ptr, (
                  subrange_field_index BaseOffsetRow))) old_builder in
189          let _ = ((Llvm.const_null base_types.int_t) $> (subrange_ptr, (
                  subrange_field_index BaseOffsetCol))) old_builder in
190          let _ = (base_var_num_rows $> (subrange_ptr, (subrange_field_index
                  SubrangeRows))) old_builder in
191          let _ = (base_var_num_cols $> (subrange_ptr, (subrange_field_index
                  SubrangeCols))) old_builder in
192          (Llvm.build_call (Hashtbl.find runtime_functions "deref_subrange_p") [|
                  subrange_ptr|] "local_id_ret_val" old_builder, old_builder) in
193        (
194          match (try StringMap.find name symbols with Not_found -> raise(LogicError("
                  Something went wrong with your semantic analysis - " ^ name ^ " not found
                  "))) with
195            LocalVariable(i) -> create_and_deref_subrange local_scope i
196          | GlobalVariable(i) -> create_and_deref_subrange global_scope i
197          | FunctionParameter(i) ->
198            let paramarray = (local_scope => (scope_field_type_index FunctionParams))
                  "paramarray" old_builder in
199            let param_addr = Llvm.build_in_bounds_gep paramarray [|Llvm.const_int
                  base_types.int_t i|] "param_addr" old_builder in
200            let param = Llvm.build_load param_addr "param" old_builder in
201            (Llvm.build_call (Hashtbl.find runtime_functions "clone_value") [|param|]
                  "function_param_ret_val" old_builder, old_builder)
202          | ExtendFunction(i) -> raise(LogicError("Something went wrong with your
                  semantic analyis - function " ^ name ^ " used as variable in RHS for " ^
                  varname))
203        )
204      | ReducedTernary(cond_var, true_var, false_var) ->
205        let get_llvm_var name getvar_builder =
206          match (try StringMap.find name symbols with Not_found -> raise(LogicError("
                  Something went wront with your transformation - Reduced Ternary name " ^
                  name ^ " not found"))) with
207            LocalVariable(i) -> Llvm.build_call getVar [|local_scope; Llvm.const_int
                  base_types.int_t i|] "llvm_var" getvar_builder
208          | GlobalVariable(i) -> Llvm.build_call getVar [|global_scope; Llvm.const_int
                  base_types.int_t i|] "llvm_var" getvar_builder
209          | _ -> raise(LogicError("Something went wront with your transformation -
                  Reduced Ternary name " ^ name ^ " not a local or global variable")) in
210
211        let (empty_bb, empty_builder) = make_block "empty" in
212        let (not_empty_bb, not_empty_builder) = make_block "not_empty" in
213        let (truthy_bb, truthy_builder) = make_block "truthy" in
214        let (falsey_bb, falsey_builder) = make_block "falsey" in
215        let (merge_bb, merge_builder) = make_block "merge" in
216
217        let ret_val_addr = Llvm.build_alloca base_types.value_p "tern_ret_val_addr"
                  old_builder in
218        let cond_llvm_var = get_llvm_var cond_var old_builder in
219        let cond_val = Llvm.build_call getVal [|cond_llvm_var; cell_row; cell_col|] "
                  cond_val" old_builder in
220        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
257          (rhs_idx_ptr, next_builder)
258        | Rel(e) ->
259          let (idx_expr_val, next_builder) = build_expr idx_builder e in
260          let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
                     next_builder in
261          let _ = (idx_expr_val $> (rhs_idx_ptr, (rhs_index_field_index RhsExprVal))
                     ) next_builder in
```

82

```
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_litrange_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
                Status))) literal_bod in
357
358          let get_val_p e = let (vp, _) = build_expr literal_bod e in vp in
359          let val_p_list_list = List.map (fun x -> List.map get_val_p x) rl in
360          let cellnums = zero_until (num_rows * num_cols) in
361          let build_empty x =
362            let emptyval = Llvm.build_malloc base_types.value_t ("" ^ (string_of_int x
                )) literal_bod in
363            let _ = store_empty emptyval literal_bod in
364            let emptydst = Llvm.build_in_bounds_gep vals_array [|Llvm.const_int
                base_types.int_t x|] "" literal_bod in
365            let _ = Llvm.build_store emptyval emptydst literal_bod in
366            let statusdst = Llvm.build_in_bounds_gep status_array [|Llvm.const_int
                base_types.int_t x|] "" literal_bod in
367            let _ = Llvm.build_store (Llvm.const_int base_types.char_t (
                var_instance_status_flags_index Calculated)) statusdst literal_bod in
368            () in
369          List.iter build_empty cellnums ;
370          let store_val r c realval =
371            let realdst = Llvm.build_in_bounds_gep vals_array [|Llvm.const_int
                base_types.int_t (r * num_cols + c)|] ("litrangeelemdst" ^ (
                string_of_int r) ^ "_" ^ (string_of_int c)) literal_bod in
372            let _ = Llvm.build_store realval realdst literal_bod in
373            () in
374          let store_row r cols = List.iteri (fun c v -> store_val r c v) cols in
375          List.iteri store_row val_p_list_list ;
376          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                initbod_val_p; Llvm.const_pointer_null base_types.char_p|] "" literal_bod
                in *)
377
378          let local_val_p = Llvm.build_load global_val_p_p "local_value_p" old_builder
                in
379          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                local_val_p; Llvm.const_pointer_null base_types.char_p|] "" old_builder
                in *)
380          let ret_val = Llvm.build_call (Hashtbl.find runtime_functions "clone_value")
                [|local_val_p|] "ret_val" old_builder in
381          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                ret_val; Llvm.const_pointer_null base_types.char_p|] "" old_builder in *)
382          (ret_val, old_builder)
383      | Call(fn,exl) -> (*TODO: Call needs to be reviewed. Possibly switch call
            arguments to value_p*)
384        let build_one_expr (arg_list, intermediate_builder) e =
385          let (arg_val, next_builder) = build_expr intermediate_builder e in
386          (arg_val :: arg_list, next_builder) in
387        let (reversed_arglist, call_builder) = List.fold_left build_one_expr ([],
            old_builder) exl in
388        let args = Array.of_list (List.rev reversed_arglist) in
389        let result = Llvm.build_call (
390          StringMap.find fn function_llvalues
391          ) args "call_ret_val" call_builder in
392        (result, call_builder)
393      | BinOp(expr1,op,expr2) -> (
394          let (val1, builder1) = build_expr old_builder expr1 in
395          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
407              let _ = Llvm.build_store
408                  (
409                    Llvm.const_int
410                    base_types.char_t
411                    (value_field_flags_index Empty)
412                  ) (
413                    Llvm.build_struct_gep
414                    ret_val
415                    (value_field_index Flags)
416                    ""
417                    int_builder
418                  )
419                  int_builder
420              in
421              let bailout = (Llvm.append_block context "" form_decl) in
422              let bbailout = Llvm.builder_at_end context bailout in
423              let (numnum_bb, numnum_builder) = make_block "numnum" in
424              let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                   numnum_builder in
425              let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                   numnum_builder in
426              let numeric_res = oppp numeric_val_1 numeric_val_2 "numeric_res"
                   numnum_builder in
427              let _ = Llvm.build_store
428                  numeric_res (
429                    Llvm.build_struct_gep
430                    ret_val
431                    (value_field_index Number)
432                    ""
433                    numnum_builder
434                  )
435                  numnum_builder in
436              let _ = Llvm.build_store
437                  (
438                    Llvm.const_int
439                    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
                       number_number "" int_builder) numnum_bb bailout int_builder in
451              (ret_val, bbailout)
452          )
453          and build_simple_int_binop oppp int_builder =
454            (let ret_val = Llvm.build_malloc base_types.value_t "binop_minus_ret_val"
                    int_builder in
455              let _ = Llvm.build_store
456                    (
457                        Llvm.const_int
458                        base_types.char_t
459                        (value_field_flags_index Empty)
460                      ) (
461                        Llvm.build_struct_gep
462                        ret_val
463                        (value_field_index Flags)
464                        ""
465                        int_builder
466                      )
467                      int_builder
468                in
469              let bailout = (Llvm.append_block context "" form_decl) in
470              let bbailout = Llvm.builder_at_end context bailout in
471              let (numnum_bb, numnum_builder) = make_block "numnum" in
472              let roundfl x = Llvm.build_call (Hashtbl.find runtime_functions "lrint
                    ") [|x|] "" numnum_builder in
473              let numeric_val_1 = roundfl ((val1 => (value_field_index Number)) "
                    number_one" numnum_builder) in
474              let numeric_val_2 = roundfl ((val2 => (value_field_index Number)) "
                    number_two" numnum_builder) in
475              let numeric_res = oppp numeric_val_1 numeric_val_2 "numeric_res"
                    numnum_builder in
476              let _ = Llvm.build_store
477                    (Llvm.build_sitofp numeric_res base_types.float_t "" numnum_builder
                        )
478                      (
479                        Llvm.build_struct_gep
480                        ret_val
481                        (value_field_index Number)
482                        ""
483                        numnum_builder
484                      )
485                      numnum_builder in
486              let _ = Llvm.build_store
487                    (
488                        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
                        number_number "" int_builder) numnum_bb bailout int_builder in
501                  (ret_val, bbailout)
502                ) in
503          let build_boolean_op numeric_comparator string_comparator int_builder =
504            let ret_val = Llvm.build_malloc base_types.value_t "binop_gt_ret_val"
                  int_builder in
505            let (make_true_bb, make_false_bb, make_empty_bb, merge_builder) =
                  make_truthiness_blocks "binop_eq" ret_val in
506
507            let (numnum_bb, numnum_builder) = make_block "numnum" in
508            let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                  numnum_builder in
509            let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                  numnum_builder in
510            let numeric_greater = Llvm.build_fcmp numeric_comparator numeric_val_1
                  numeric_val_2 "numeric_greater" numnum_builder in
511            let _ = Llvm.build_cond_br numeric_greater make_true_bb make_false_bb
                  numnum_builder in
512
513            let (strstr_bb, strstr_builder) = make_block "strstr" in
514            let str_p_1 = (val1 => (value_field_index String)) "string_one"
                  strstr_builder in
515            let str_p_2 = (val2 => (value_field_index String)) "string_two"
                  strstr_builder in
516            let char_p_1 = (str_p_1 => (string_field_index StringCharPtr)) "char_p_one
                  " strstr_builder in
517            let char_p_2 = (str_p_2 => (string_field_index StringCharPtr)) "char_p_two
                  " strstr_builder in
518            let strcmp_result = Llvm.build_call (Hashtbl.find runtime_functions "
                  strcmp") [|char_p_1; char_p_2|] "strcmp_result" strstr_builder in
519            let string_greater = Llvm.build_icmp string_comparator strcmp_result (Llvm
                  .const_null base_types.long_t) "string_greater" strstr_builder in
520            let _ = Llvm.build_cond_br string_greater make_true_bb make_false_bb
                  strstr_builder in
521
522            let switch_inst = Llvm.build_switch combined_type make_empty_bb 2
                  int_builder in (* Incompatible ===> default to empty *)
523            Llvm.add_case switch_inst number_number numnum_bb;
524            Llvm.add_case switch_inst string_string strstr_bb;
525            (ret_val, merge_builder) in
526          match op with
527            Minus -> build_simple_binop Llvm.build_fsub int_builder
528          | Plus ->
529              let result = Llvm.build_malloc base_types.value_t "" int_builder
```

89

```
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 numorstrorother = (Llvm.append_block context "" form_decl)
534                  and strorother = (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 bnumorstrorother = Llvm.builder_at_end context numorstrorother
539                  and bstrorother = Llvm.builder_at_end context strorother
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) "" bnumorstrorother)
                         "" bnumorstrorother) (Llvm.const_int base_types.char_t (
                         value_field_flags_index Number)) "" bnumorstrorother
544                  and isstring = Llvm.build_icmp Llvm.Icmp.Eq (Llvm.build_load (Llvm.
                         build_struct_gep val1 (value_field_index Flags) "" bstrorother) ""
                         bstrorother) (Llvm.const_int base_types.char_t (
                         value_field_flags_index String)) "" bstrorother
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) "" 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 (
```

```
                              value_field_index String) "" bstradd) bstradd in
558            let fullLen = Llvm.build_nsw_add (Llvm.build_nsw_add len1 len2 ""
                  bstradd) (Llvm.const_int base_types.long_t 1) "" bstradd
559            and extra_byte2 = (Llvm.build_add len2 (Llvm.const_int base_types.long_t
                   1) "" bstradd) in
560            let dst_char = Llvm.build_array_malloc base_types.char_t (Llvm.
                  build_trunc fullLen base_types.int_t "" bstradd) "" bstradd in
561            let dst_char2 = Llvm.build_in_bounds_gep dst_char [|len1|] "" bstradd in
562            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
563            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
564            and _ = Llvm.build_store dst_char dst_char_ptr_ptr bstradd
565            in
566            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
567            in
568            let _ = Llvm.build_cond_br isnumorstring numorstrorother bailout
                  int_builder
569            and _ = Llvm.build_cond_br isnumber numadd strorother bnumorstrorother
570            and _ = Llvm.build_cond_br isstring stradd bailout bstrorother
571            and _ = Llvm.build_br bailout bstradd
572            and _ = Llvm.build_br bailout bnumadd
573            in
574            (result, bbailout)
575        | Times −> build_simple_binop Llvm.build_fmul int_builder
576        | Eq −>
577          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print")
                  [|val1; Llvm.build_global_stringptr "Eq operator − value 1" ""
                  old_builder|] "" int_builder in
578          let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                  val2; Llvm.build_global_stringptr "Eq operator − value 2" ""
                  old_builder|] "" int_builder in *)
579          let ret_val = Llvm.build_malloc base_types.value_t "binop_eq_ret_val"
                  int_builder in
580          let (make_true_bb, make_false_bb, _, merge_builder) =
                  make_truthiness_blocks "binop_eq" ret_val in
581
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
588          let (strstr_bb, strstr_builder) = make_block "strstr" in
589          let str_p_1 = (val1 => (value_field_index String)) "string_one"
                  strstr_builder in
590          let str_p_2 = (val2 => (value_field_index String)) "string_two"
```

91

```
                                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"
```

95

```
                          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. TOOD: 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 expresssions 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
```

98

```
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
873          LocalVariable(i) ->
874            let llvm_var = Llvm.build_call getVar [|scope_obj; Llvm.const_int
                 base_types.int_t i|] (special_name ^ "_var") builder in
```

```
875          Llvm.build_call getVal [|llvm_var; Llvm.const_int base_types.int_t 0; Llvm
                 .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 _ = Llvm.build_call (Hashtbl.find runtime_functions "verify_assert") [|
               assert_val; Llvm.build_global_stringptr fname "" builder|] "" builder in
880     let ret_val = get_special_val "return" (snd fn_def.func_ret_val) in
881     let _ = Llvm.build_ret ret_val builder in () in
882   (* End of build_function *)
883
884   (* Build the global scope object *)
885   let vardefn_p_p = Llvm.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 _ = Llvm.build_call (Hashtbl.find runtime_functions "incStack") [||] "" init_bod
           in
888   let _ = Llvm.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 _ = Llvm.build_ret_void init_bod in
896   let _ = Llvm.build_ret_void literal_bod in
897   let _ = Llvm.build_call init_def [||] "" main_bod in
898   let _ = Llvm.build_call literal_def [||] "" main_bod in
899   let cmd_line_args = Llvm.build_call (Hashtbl.find runtime_functions "
             box_command_line_args") [|Llvm.param main_def 0; Llvm.param main_def 1|] "
             cmd_line_args" main_bod in
900   let _ = Llvm.build_call extend_entry_point [|cmd_line_args|] "" main_bod in
901   let _ = Llvm.build_ret (Llvm.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 _ = Llvm_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(); // asssuming 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    int fileNum = (int) file_handle->numericVal;
172    if (fileNum > open_num_files || open_files[fileNum] == NULL) {
173      // Per the LRM this is actually supposed to crash the program.
174      fprintf(stderr, "EXITING - Attempted to write to something that was not a valid
             file pointer\n");
175      exit(-1);
176    }
177    fwrite(buffer->str->text, 1, buffer->str->length, open_files[fileNum]);
178    // TODO: make this return empty once compiler handles Id(s)
179    // RN: Use the return value to close the file
180    return new_number((double) fileNum);
181  }
182
183  #ifdef PLOT
184  value_p extend_plot(value_p file_name){
185    // extract the numerical values from the first parameter - values
186    if(!assertSingle(file_name)) return new_val();
187    float a[6]  = { 0.5, 0.09, 0.6, 0.85, 0.0, 0.90 },
188          b[6]  = { 1.9, 1.3,  0.6, 0.75, 0.1, 2.0 };
189    char *t[6] = { "Chicago", "New York", "L.A.", "Atlanta", "Paris, MD\n(USA) ", "
             London" };
190    unsigned long sc[2]    = { 0xFF8080, 0x8080FF };
191    GDC_BGColor   = 0xFFFFFFL;
192    GDC_LineColor = 0x000000L;
193    GDC_SetColor  = &(sc[0]);
194    GDC_stack_type = GDC_STACK_BESIDE;
195    // Using the line below, can also spit to stdout and fwrite from Extend
196    // printf( "Content-Type: image/png\n\n" );
197    FILE *outpng = fopen("extend.png", "wb");
198    out_graph(250, 200, outpng, GDC_3DBAR, 6, t, 2, a, b);
199    fclose(outpng);
200    return new_val();
201  }
202
203  value_p extend_bar_chart(value_p file_handle, value_p labels, value_p values){
204    // Mandates 1 row, X columns
205    if(!assertSingleNumber(file_handle)) return new_val();
206    int fileNum = (int)file_handle->numericVal;
207    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  }
326
327  value_p extend_random() {
328          return new_number(rand());
329  }
```

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",
        which_value == NULL ? "some anonymous variable" : which_value);
23    fprintf(stderr, "Memory address: %p\n", val);
24    if (val == NULL) {
25      fprintf(stderr, "—————————Nice try asking me to dereference a null pointer\n
            —————————");
26      return;
27    }
28    fprintf(stderr, "Flags: %d (%s)\n", val->flags, flag_meanings[val->flags]);
29    fprintf(stderr, "NumericVal: %f\n", val->numericVal);
30    fprintf(stderr, "String contents: Probably safer not to check that pointer (%p)
        blindly\n", val->str);
31    if (val->flags == FLAG_STRING && val->str != NULL) {
32      fprintf(stderr, "It says it's a string and it's not a NULL pointer though, so here
            you go:\n");
33      fprintf(stderr, "String refcount: %d\n", val->str->refs);
34      fprintf(stderr, "String length: %ld\n", val->str->length);
35      fprintf(stderr, "String char* memory address: %p\n", val->str->text);
36      if (val->str->text == NULL) {
37        fprintf(stderr, "Not going to print the contents of NULL!\n");
```

```c
38      } else {
39        fprintf(stderr, "String char* contents:\n%s\n", val->str->text);
40      }
41    }
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     exit(-1);
110   }
111   fprintf(stderr, "Index type: ");
112   switch(idx->rhs_index_type) {
113     case RHS_IDX_ABSOLUTE:
114       fprintf(stderr, "Absolute\n");
115       if (idx->val_of_expr == NULL) {
116         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");
117       } else {
118         debug_print(idx->val_of_expr, "an absolute index");
119       }
120       break;
121     case RHS_IDX_RELATIVE:
122       fprintf(stderr, "Relative\n");
123       if (idx->val_of_expr == NULL) {
124         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");
125       } else {
126         debug_print(idx->val_of_expr, "a relative index");
127       }
128       break;
129     case RHS_IDX_DIM_START:
130       fprintf(stderr, "DimensionStart\n");
131       if (idx->val_of_expr != NULL) {
132         fprintf(stderr, "This definitely isn't supposed to happen — the value pointer
                isn't NULL. You should look into that.\n");
133         exit(-1);
```

```
134          }
135        break;
136      case RHS_IDX_DIM_END:
137        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->slice1, sel->slice2);
177    if (sel->slice1 != NULL) {
178      fprintf(stderr, "Slice 1 info:\n");
179      debug_print_slice(sel->slice1);
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
```

```c
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
               way.\n", scope_ptr->defns[varnum].name);
310      exit(-1);
311    }
312    struct var_instance *inst = get_variable(scope_ptr, varnum);
313    if (inst->rows != 1 || inst->cols != 1) {
314      fprintf(stderr, "A variable (%s) that is defined as one by one is somehow actually
               %d by %d.\n", inst->name, inst->rows, inst->cols);
315      exit(-1);
316    }
317    value_p val = getVal(inst, 0, 0);
318    if (!assertSingleNumber(val) || !isfinite(val->numericVal)) {
319      return 0;
320    }
321    *result = (int) lrint(val->numericVal);
322    return 1;
323  }
324
325  struct var_instance *instantiate_variable(struct ExtendScope *scope_ptr, struct
       var_defn def) {
326    struct var_instance *inst = malloc(sizeof(struct var_instance));
327    if(def.isOneByOne) {
328      inst->rows = 1;
329      inst->cols = 1;
330    } else {
331      if (!getIntFromOneByOne(scope_ptr, def.rows_varnum, &inst->rows)) {
332        fprintf(stderr, "EXITING - The expression for the number of rows of variable %s
                 did not evaluate to a finite Number.\n", def.name);
333        exit(-1);
334      }
335      if (!getIntFromOneByOne(scope_ptr, def.cols_varnum, &inst->cols)) {
336        fprintf(stderr, "EXITING - The expression for the number of columns of variable
                 %s did not evaluate to a finite Number.\n", def.name);
337        exit(-1);
338      }
339      if (inst->rows <= 0 || inst->cols <= 0) {
340        fprintf(stderr, "EXITING - The requested dimensions for variable %s were [%d, %d
                 ]; they must both be greater than zero.\n", def.name, inst->rows, inst->cols)
                 ;
341        exit(-1);
342      }
```

114

```
343     }
344     // TODO: do the same thing for each FormulaFP to turn an ExtendFormula into a
            ResolvedFormula
345     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     return scope_ptr->vars[varnum];
452   }
453
454   char assertInBounds(struct var_instance *defn, int r, int c) {
455     return (
456       r >= 0 && r < defn->rows &&
457       c >= 0 && c < defn->cols
458     );
459   }
460
461   value_p calcVal(struct var_instance *inst, int r, int c) {
462     int i;
463     for (i = 0; i < inst->numFormulas; i++) {
464       if (
465         r >= inst->formulas[i].rowStart && r < inst->formulas[i].rowEnd &&
466         c >= inst->formulas[i].colStart && c < inst->formulas[i].colEnd
467       ) {
468         return (inst->formulas[i].formula)(inst->closure, r, c);
469       }
470     }
471     return new_val();
472   }
473
474   value_p clone_value(value_p old_value) {
475     value_p new_value = (value_p) malloc(sizeof(struct value_t));
476     new_value->flags = old_value->flags;
477     switch (new_value->flags) {
478       case FLAG_EMPTY:
479         break;
480       case FLAG_NUMBER:
481         new_value->numericVal = old_value->numericVal;
482         break;
483       case FLAG_STRING:
484         new_value->str = old_value->str;
485         new_value->str->refs++;
486         break;
487       case FLAG_SUBRANGE:
488         new_value->subrange = (subrange_p) malloc(sizeof(struct subrange_t));
489         memcpy(new_value->subrange, old_value->subrange, sizeof(struct subrange_t));
490         if (new_value->subrange->range->closure != NULL) {
491           new_value->subrange->range->closure->refcount++; /* Not sure about this one */
```

```
492          }
493        break;
494      default:
495        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
                 ref count now */
524        break;
525      case FLAG_SUBRANGE:
526        delete_subrange_p(old_value->subrange);
527        break;
528      default:
529        fprintf(stderr, "delete_value(%p): Illegal value of flags: %c\n", old_value,
                 old_value->flags);
530        exit(-1);
531        break;
532    }
533 }
534
535 value_p deref_subrange_p(subrange_p subrng) {
536    if (subrng == NULL) {
537      fprintf(stderr, "Exiting - asked to dereference a NULL pointer.\n");
538      exit(-1);
539    }
540    if (subrng->subrange_num_rows == 1 && subrng->subrange_num_cols == 1) {
541      return getVal(subrng->range, subrng->base_var_offset_row, subrng->
                 base_var_offset_col);
542    } else {
543      value_p new_value = (value_p) malloc (sizeof(struct value_t));
```

```
544      new_value->flags = FLAG_SUBRANGE;
545      new_value->numericVal = 0.0;
546      new_value->str = NULL;
547      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;
```

121

```
708          col_slice_p = &zero_to_one;
709        } else {
710          return new_val();
711  /*       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           exit(-1);
790           break;
791         case CALCULATED:
792           if (inst->values[cell_number] == NULL) {
793             fprintf(stderr, "Supposedly, %s[%d,%d] was already calculated, but there is a
                       null pointer there.\n", inst->name, r, c);
794             fprintf(stderr, "Attempting to print contents of the variable instance where
                       this occurred:\n");
795             fflush(stderr);
796             debug_print_varinst(inst);
797             exit(-1);
798           }
799           break;
800         default:
801           fprintf(stderr, "Unrecognized cell status %d (row %d, col %d)!\n", cell_status,
                     r, c);
802           fprintf(stderr, "Attempting to print contents of the variable instance where
                     this occurred:\n");
803           fflush(stderr);
804           debug_print_varinst(inst);
805           exit(-1);
806           break;
807       }
808       return inst->values[cell_number];
809     }
```

9.12   stdlib.xtnd

```
 1  /* jss2272 ns3158 isk2108 */
 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    random(); // Just for fun − very non−random.
42  }
43
44  global STDIN := get_stdin();
45  global STDOUT := get_stdout();
46  global STDERR := get_stderr();
47
48  print_endline(val) {
49    return write(STDOUT, toString(val) + "\n");
50  }
51
52  transpose([m,n] rng) {
53    [n,m] ret := rng[column(),row()];
```

```
54     return ret;
55   }
56
57   flatten([m,n] rng) {
58     [1,m*n] ret := rng[floor(column()/n), column()%n];
59     return ret;
60   }
61
62   isNumber(x) {
63     return typeof(x) == "Number";
64   }
65
66   isEmpty(x) {
67     return typeof(x) == "Empty";
68   }
69
70   colRange(start, end) {
71     [end-start, 1] ret;
72     ret[0,0] = start;
73     ret[1:,0] = ret[[-1]] + 1;
74     return ret;
75   }
76
77   rowRange(start, end) {
78     return transpose(colRange(start,end));
79   }
80
81   matchCol([num_rows, 1] list, val) {
82     [num_rows, 1] amt_to_add, final_index;
83     amt_to_add[0,0] = val == #list ? 0 : 1;
84     amt_to_add[1:,0] = (amt_to_add[[-1]] == 0 || val == #list) ? 0 : 1;
85     final_index[0,0] = 0;
86     final_index[1:,0] = final_index[[-1]] + amt_to_add[[-1]];
87     return amt_to_add[-1] == 0 ? final_index[-1] : empty;
88   }
89
90   matchRow([1, num_cols] list, val) {
91     [1, num_cols] amt_to_add, final_index;
92     amt_to_add[0,0] = val == #list ? 0 : 1;
93     amt_to_add[0,1:] = (amt_to_add[[-1]] == 0 || val == #list) ? 0 : 1;
94     final_index[0,0] = 0;
95     final_index[0,1:] = final_index[[-1]] + amt_to_add[[-1]];
96     return amt_to_add[-1] == 0 ? final_index[-1] : empty;
97   }
98
99   match([m,n] list, val) {
100    return m == 1 ? matchRow(list, val) : (n == 1 ? matchCol(list, val) : empty);
101  }
102
103  bsearch([num_rows, 1] list, val) {
104    mid := (num_rows - 1) / 2;
105    return switch {
106      case list[mid] == val:
107        mid;
108      case list[mid] > val:
109        mid > 0 ? bsearch(list[:mid], val) : empty;
```

```
110      case list[mid] < val:
111        num_rows > 1 ? mid + 1 + bsearch(list[mid+1:], val) : empty;
112    };
113  }
114
115  sum_column([m,1] rng) {
116    [m,1] running_sum;
117    running_sum[0,0] = #rng;
118    running_sum[1:,0] = running_sum[[-1]] + #rng;
119    return running_sum[-1];
120  }
121
122  sum([m,n] rng) {
123    /* Returns the sum of the values in the range, skipping any values that are non-
           numeric */
124    [m,n] numbers := isNumber(#rng) ? #rng : 0;
125    [1,n] column_sums := sum_column(numbers[:,]);
126    return sum_column(transpose(column_sums));
127  }
128
129  nmax(n1, n2) {
130    return n1 > n2 ? n1 : n2;
131  }
132
133  max_column([m,1] rng) {
134    [m,1] running_max;
135    running_max[0,0] = #rng;
136    running_max[1:,0] = running_max[[-1]] > #rng ? running_max[[-1]] : #rng;
137    return running_max[-1];
138  }
139
140  max([m,n] rng) {
141    /* Returns the max of the values in the range, skipping any values that are non-
           numeric */
142    [m,n] numbers := isNumber(#rng) ? #rng : empty;
143    [1,n] column_maxs := max_column(rng[:,]);
144    return max_column(transpose(column_maxs));
145  }
146
147  nmin(n1, n2) {
148    return n1 < n2 ? n1 : n2;
149  }
150
151  min_column([m,1] rng) {
152    [m,1] running_min;
153    running_min[0,0] = #rng;
154    running_min[1:,0] = running_min[[-1]] > #rng ? running_min[[-1]] : #rng;
155    return running_min[-1];
156  }
157
158  min([m,n] rng) {
159    /* Returns the min of the values in the range, skipping any values that are non-
           numeric */
160    [m,n] numbers := isNumber(#rng) ? #rng : empty;
161    [1,n] column_mins := min_column(rng[:,]);
162    return min_column(transpose(column_mins));
```

```
163  }
164
165  sign(arg) {
166    return switch {
167      case arg > 0: 1;
168      case arg < 0: −1;
169      case arg == 0: 0;
170    };
171  }
172
173  gcd(m, n) {
174    return (n == 0) ? m : gcd(n, m % n);
175  }
176
177  lcm(m, n) {
178    return m * n / gcd(m, n);
179  }
180
181  sumsq([m,n] rng) {
182    [m,n] squares := #rng * #rng;
183    return sum(squares);
184  }
185
186  sumproduct([m,n] rng1, [m,n] rng2) {
187    [m,n] products := #rng1 * #rng2;
188    return sum(products);
189  }
190
191  sumxmy2([m,n] rng1, [m,n] rng2) {
192    [m,n] diffs := #rng1 − #rng2;
193    return sumsq(diffs);
194  }
195
196  mmult([m,n] rng1, [n,p] rng2) {
197    [m,p] result := sumproduct(rng1[,:],transpose(rng2[:,]));
198    return result;
199  }
200
201  linest([p,q] known_ys, [p,q] known_xs) {
202    flat_ys := flatten(known_ys);
203    flat_xs := flatten(known_xs);
204
205    n := p * q;
206    S_x := sum(flat_xs);
207    S_y := sum(flat_ys);
208    S_xx := sumsq(flat_xs);
209    S_yy := sumsq(flat_ys);
210    S_xy := sumproduct(flat_xs, flat_ys);
211
212    beta1_hat := (n * S_xy − S_x*S_y)/(n*S_xx − S_x*S_x);
213    beta0_hat := S_y / n − beta1_hat * S_x / n;
214    [2,2] ret;
215    ret[0,0] = "Intercept estimate";
216    ret[0,1] = "Slope estimate";
217    ret[1,0] = beta0_hat;
218    ret[1,1] = beta1_hat;
```

```
219   return ret;
220 }
221
222 toUpper(text) {
223   val := toASCII(text);
224   val_s := size(val);
225   [val_s[0],val_s[1]] result := #val >= 97 && #val <= 122 ? #val − 32 : #val;
226   return fromASCII(result);
227 }
228
229 toLower(text) {
230   val := toASCII(text);
231   val_s := size(val);
232   [val_s[0],val_s[1]] result := #val >= 65 && #val <= 90 ? #val + 32 : #val;
233   return fromASCII(result);
234 }
235
236 left(str, num_chars) {
237   return fromASCII(toASCII(str)[:num_chars]);
238 }
239
240 right(str, num_chars) {
241   return fromASCII(toASCII(str)[−num_chars:]);
242 }
243
244 substring(str, start, length) {
245   return fromASCII(toASCII(str)[start:start+length]);
246 }
247
248 concatRow([1,n] cells, joiner) {
249   [1,n] accum, strings := toString(#cells);
250   accum[0,0] = #strings;
251   accum[0,1:] = accum[[−1]] + joiner + #strings;
252   return accum[−1];
253 }
254
255 concatCol([m,1] cells, joiner) {
256   return transpose(concatRow(transpose(cells), joiner));
257 }
258
259 join([m,n] cells, joiner) {
260   return m == 1 ? concatRow(cells, joiner) : (n == 1 ? concatCol(cells, joiner) :
        empty);
261 }
262
263 joinRange([m,n] cells, rowJoiner, colJoiner) {
264   [m,n] strings := toString(#cells);
265   [m,1] joinedRows := concatRow(strings[,:], colJoiner);
266   return concatCol(joinedRows, rowJoiner);
267 }
268
269 toRangeLiteral([m,n] rng) {
270   [m,n] strings := toLiteral(#rng);
271   [m,1] rows := concatRow(strings[,:], ", ");
272   return "{" + concatRow(transpose(rows), ";\n") + "}";
273 }
```

```
274
275  toLiteral(arg) {
276    return switch(typeof(arg)) {
277      case "Number":
278        toString(arg);
279      case "String":
280        "\"" + arg + "\"";
281      case "Empty":
282        "empty";
283      case "Range":
284        toRangeLiteral(arg);
285    };
286  }
287
288  repeat(str, num) {
289    [1,num] copies := str;
290    return concatRow(copies,"");
291  }
292
293  stringOfPositiveInteger(arg) {
294    num_digits := 1 + floor(log10(arg));
295    [1,num_digits] digits := floor(arg/10**(num_digits—1—column())) % 10;
296    [1,num_digits] ascii_digits := 48 + #digits;
297    return arg < 1 ? "0" : fromASCII(ascii_digits);
298  }
299
300  padLeft(str, pad_char, total_length) {
301    existing_length := len(str);
302    padding := repeat(pad_char, total_length — len(str));
303    return existing_length < total_length ? (padding + str) : str;
304  }
305
306  toString(arg) {
307    positive_arg := fabs(arg);
308    closest_integer := round(positive_arg, 0);
309    is_integral_enough := fabs(positive_arg—closest_integer) < rounding_cutoff;
310    floating_part := round(10 ** digits_after_decimal * (positive_arg — floor(
           positive_arg)),0);
311    positive_part := stringOfPositiveInteger(floor(positive_arg)) + (is_integral_enough
           ? "" : "." + padLeft(stringOfPositiveInteger(floating_part), "0",
           digits_after_decimal));
312
313    return switch(typeof(arg)) {
314      case "Number":
315        switch {
316          case isNaN(arg):
317            "NaN";
318          case isInfinite(arg) == —1:
319            "—Inf";
320          case isInfinite(arg) == 1:
321            "Inf";
322          case sign(arg) == 0:
323            "0";
324          case sign(arg) == 1:
325            positive_part;
326          case sign(arg) == —1:
```

```
327              "−" + positive_part;
328          default:
329            "Encountered a number that is neither NaN, +Inf, −Inf, 0, positive or
                  negative";
330        };
331      case "String":
332        arg;
333      case "Empty":
334        "empty";
335      case "Range":
336        toRangeLiteral(arg);
337    };
338  }
339
340  numRows(arg) {
341    return size(arg)[0];
342  }
343
344  numCols(arg) {
345    return size(arg)[1];
346  }
347
348  splitChars([1,n] stringchars, splitchar) {
349    loc := matchRow(stringchars, splitchar);
350    firstword := fromASCII(stringchars[:loc]);
351    lastwords := splitChars(stringchars[loc+1:],splitchar);
352    combined := stack(firstword, lastwords);
353    return loc == empty ? fromASCII(stringchars) : combined;
354  }
355
356  split(string, splitter) {
357    return splitChars(toASCII(string), toASCII(splitter));
358  }
359
360  splitToRange(string, row_splitter, col_splitter) {
361    split_rows := split(string, row_splitter);
362    [numRows(split_rows),1] split_cols := split(#split_rows,col_splitter);
363    [numRows(split_rows),1] col_lengths := numRows(#split_cols);
364    [numRows(split_rows), max(col_lengths)] result := #split_cols[column()];
365    return result;
366  }
367
368  isSpace(char) {
369    return switch(char) {
370      case toASCII(" "), toASCII("\n"), toASCII("\t"), toASCII("\r"):
371        1;
372      default:
373        0;
374    };
375  }
376
377  trimChars(chars) {
378    return isSpace(chars[0]) ? trimChars(chars[1:]) : chars;
379  }
380
381  ltrim(s) {
```

```
382     return fromASCII(trimChars(toASCII(s)));
383  }
384
385  reverse(s) {
386    chars := toASCII(s);
387    l := len(s);
388    [1,numCols(chars)] chars_reversed := chars[l−1−column()];
389    return l ? fromASCII(chars_reversed) : "";
390  }
391
392  rtrim(s) {
393    return reverse(ltrim(reverse(s)));
394  }
395
396  trim(s) {
397    return ltrim(rtrim(s));
398  }
399
400  charAt(s, i) {
401    return toASCII(s)[i];
402  }
403
404  parseString(s) {
405    trimmed := trim(s);
406    rangeSplit := splitToRange(substring(trimmed, 1, len(trimmed) − 2), ";", ",");
407    [numRows(rangeSplit), numCols(rangeSplit)] rangeContents := parseString(#rangeSplit)
            ;
408    return switch {
409      case charAt(trimmed,0) == toASCII("{") && charAt(trimmed,−1) == toASCII("}"):
410        rangeContents;
411      case charAt(trimmed,0) == toASCII("\"") && charAt(trimmed,−1) == toASCII("\""):
412        substring(trimmed, 1, len(trimmed) − 2);
413      case trimmed == "empty":
414        empty;
415      default:
416        parseFloat(trimmed);
417    };
418  }
419
420  normalize([m,n] arg) {
421    [m,n] squared_lengths := #arg * #arg, normalized := #arg / vector_norm;
422    vector_norm := sqrt(sum(squared_lengths));
423    return normalized;
424  }
425
426  append([m,n] rg1, [p,q] rg2) {
427    [nmax(m,p), n+q] res;
428    res[:m,:n] = #rg1;
429    res[:p,n:n+q] = rg2[,[−n]];
430    return res;
431  }
432
433  stack(rg1, rg2) {
434    return transpose(append(transpose(rg1), transpose(rg2)));
435  }
436
```

```
437  avg([m,n] rng) {
438     return sum(rng) / (m*n);
439  }
440
441  stdev([m,n] rng) {
442     mean := avg(rng);
443     [m,n] devs := #rng − mean;
444     return sqrt(sumsq(devs)/(m*n−1));
445  }
446
447  mergesort([m,n]data, sort_col) {
448     breakpoint := floor(m/2);
449     split1 := data[:breakpoint,:];
450     split2 := data[breakpoint:,:];
451     sizesplit1 := size(split1)[0];
452     sizesplit2 := size(split2)[0];
453     sortedsplit1 := sizesplit1 < 2 ? split1[row(),:] : mergesort(sort_col,split1);
454     sortedsplit2 := sizesplit2 < 2 ? split2[row(),:] : mergesort(sort_col,split2);
455     [m, 3] merged;
456     merged[0,0] = #isbigger ? sortedsplit1[0,:] : sortedsplit2[0,:];
457     merged[0,1] = #isbigger ? 1 : 0;
458     merged[0,2] = #isbigger ? 0 : 1;
459     [m, 1]isbigger;
460     isbigger[0,0] = sortedsplit1[0,sort_col] <= sortedsplit2[0,sort_col];
461     isbigger[1:,0] = sortedsplit1[merged[[−1], 1],sort_col] == empty ? 0 :
462       (sortedsplit2[merged[[−1],2],sort_col] == empty ? 1 :
463       (sortedsplit1[merged[[−1],1],sort_col] <= sortedsplit2[merged[[−1],2],sort_col]));
464     merged[1:,0] =
465       #isbigger
466       ? sortedsplit1[merged[[−1],1],:]
467       : sortedsplit2[merged[[−1],2],:];
468     merged[1:,1] = #isbigger ? merged[[−1],1] + 1 : merged[[−1],1];
469     merged[1:,2] = #isbigger ? merged[[−1],2] : merged[[−1],2] + 1;
470     [m, n]transfer := merged[row(),0][column()];
471     return transfer;
472  }
```

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

141

```
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
```

test-parse-error-comment.xtnd

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

test-parse-error-comment.xtnd - Expected Output

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

test-parse-error-missing-semicolon.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-parse-error-missing-semicolon.xtnd - Expected Output

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

test-parse-error-newlines.xtnd

```
1  main(args){
2    foo := 5/5;
3    bar := "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"$5;
4    return foo;
5  }
```

test-parse-error-newlines.xtnd - Expected Output

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

test-parse-error-string.xtnd

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

test-parse-error-string.xtnd - Expected Output

```
1  Syntax error in "./testcases/inputs_regression/test_parse_error_string.xtnd": Invalid
       character: $
2  Line 2 at character 18
```

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  f371045 2016—12—20T10:50:09—05:00 GitHub: Merge pull request #139 from ExtendLang/
       finalfinalfinalLRM
 2  b05d682 2016—12—20T07:49:45—05:00 oracleofnj: Rebuild LRM.pdf
 3  a655b31 2016—12—20T07:48:35—05:00 oracleofnj: Really final I hope
 4  5b5f9e4 2016—12—20T07:19:19—05:00 oracleofnj: Final stdlib and LRM
 5  f113d6d 2016—12—20T06:31:52—05:00 oracleofnj: Finally put in actual selection examples
 6  664fc38 2016—12—20T05:59:43—05:00 oracleofnj: Join
 7  4146159 2016—12—20T05:52:00—05:00 oracleofnj: Merge branch 'imp—sort' into final—
       stdlib
 8  4799835 2016—12—20T05:50:50—05:00 oracleofnj: Merge branch 'imp—sort' into final—
       stdlib
 9  d0547d2 2016—12—20T05:24:03—05:00 oracleofnj: I took a shot at it
10  edf4760 2016—12—20T00:37:01—05:00 oracleofnj: Last of stdlib
11  a50124a 2016—12—19T23:09:25—05:00 Nigel Schuster: Reduced sample, printing
12  3842c72 2016—12—19T23:07:19—05:00 Nigel Schuster: Sorting demonstrated
13  5e3e990 2016—12—19T21:40:42—05:00 Nigel Schuster: Improved sorting algorithm
14  a6db867 2016—12—19T20:36:52—05:00 GitHub: Merge pull request #135 from ExtendLang/
       contrib
15  900411c 2016—12—19T17:40:28—05:00 GitHub: Merge branch 'master' into contrib
16  d253e33 2016—12—19T17:39:19—05:00 Nigel Schuster: Assigned contributions
17  2569ea5 2016—12—19T12:43:20—05:00 GitHub: Merge pull request #134 from ExtendLang/
       stdlib—additions
18  47f1167 2016—12—19T12:09:58—05:00 GitHub: Merge branch 'master' into stdlib—additions
19  2e18028 2016—12—19T12:09:50—05:00 GitHub: Merge pull request #133 from ExtendLang/
       mergesort
20  142938e 2016—12—19T11:51:25—05:00 oracleofnj: Couple stdlib additions
21  3fc2843 2016—12—19T10:17:09—05:00 Nigel Schuster: Mergesort example
22  0b23496 2016—12—19T09:07:15—05:00 GitHub: Merge pull request #132 from ExtendLang/
       fixing—tcs
23  b86d992 2016—12—19T00:38:27—05:00 oracleofnj: Verify samples, compact TSP
24  e859c75 2016—12—19T00:17:57—05:00 oracleofnj: Final test run with complete stdlib
25  f02f2da 2016—12—19T00:13:20—05:00 oracleofnj: Remove obselete print flavors
26  dd65154 2016—12—19T00:08:17—05:00 oracleofnj: Merge branch 'master' into fixing—tcs
27  82f4ad5 2016—12—19T00:07:45—05:00 oracleofnj: Merge branch 'master' of https://github.
       com/ExtendLang/Extend
28  d2cd19e 2016—12—19T00:07:30—05:00 oracleofnj: 125 / 125
29  8d02537 2016—12—19T00:07:11—05:00 GitHub: Merge pull request #131 from ExtendLang/
       webgif
30  453b7f6 2016—12—18T23:33:06—05:00 oracleofnj: Back to passing all previously passing
       TCs; on to stragglers
31  6487840 2016—12—18T23:00:01—05:00 Nigel Schuster: Removed webgif from Makefile
32  84dca34 2016—12—18T22:29:59—05:00 oracleofnj: Ignore webgif
33  ed204c2 2016—12—18T22:29:32—05:00 oracleofnj: Ignore webgif
34  1bcb830 2016—12—18T19:48:02—05:00 GitHub: Merge pull request #41 from ExtendLang/
```

```
        plotting
35  1cd2360 2016—12—18T19:46:21—05:00 GitHub: Merge branch 'master' into plotting
36  0058659 2016—12—18T19:30:35—05:00 GitHub: Merge pull request #129 from ExtendLang/
        remove—debug—final
37  b54f4aa 2016—12—18T18:54:47—05:00 GitHub: Fix MAXFLOAT
38  9324fb8 2016—12—18T18:50:59—05:00 GitHub: Merge branch 'master' into plotting
39  f152bc9 2016—12—18T18:32:21—05:00 oracleofnj: Remove Debug()
40  c069630 2016—12—18T18:15:40—05:00 Nigel Schuster: Linking plotter is optional
41  e9dbd0f 2016—12—18T17:05:54—05:00 GitHub: Merge pull request #128 from ExtendLang/back
        —to—parsing
42  b602263 2016—12—18T14:12:48—05:00 oracleofnj: Merge in cool program
43  45d14cf 2016—12—18T14:12:28—05:00 GitHub: Merge pull request #127 from ExtendLang/
        strcat—bug
44  790bc51 2016—12—18T14:00:48—05:00 oracleofnj: Merge branch 'cool_program' into back—to
        —parsing
45  2eba5e5 2016—12—18T13:59:44—05:00 oracleofnj: Replace C extend_parseString with in—
        language parseString
46  1b664be 2016—12—18T09:14:51—05:00 Nigel Schuster: Corrected travis file
47  554b584 2016—12—18T09:13:54—05:00 Nigel Schuster: Cleand up Makefile mess
48  a222916 2016—12—18T08:19:54—05:00 GitHub: Merge branch 'master' into plotting
49  d064d8a 2016—12—18T01:49:39—05:00 Ishaan: Cleanup line function
50  47dace5 2016—12—18T01:48:28—05:00 Ishaan: Test single parameter line chart
51  8f5cf52 2016—12—18T01:43:49—05:00 Ishaan: Fix the testcase fail
52  84cd775 2016—12—18T01:41:50—05:00 Ishaan: update testcase
53  a425775 2016—12—18T01:39:43—05:00 Ishaan: Figure out 2 line issue
54  ba2c3c1 2016—12—18T01:34:34—05:00 Ishaan: Add y values and update testcase
55  7ad5986 2016—12—18T01:18:28—05:00 Ishaan: Trying another version of line
56  b8732dd 2016—12—18T00:42:25—05:00 Ishaan: Fix derp in linechart
57  20e2c43 2016—12—18T00:40:22—05:00 Ishaan: Added basic linechart function to examine
58  b404e12 2016—12—17T23:56:16—05:00 Ishaan: Cast to float
59  e866f68 2016—12—17T23:38:40—05:00 Ishaan: Reverse row and col
60  8419510 2016—12—17T23:27:13—05:00 oracleofnj: That's a wrap
61  6ec3e0e 2016—12—17T23:17:46—05:00 oracleofnj: Proof of concept
62  302af00 2016—12—17T23:09:14—05:00 Ishaan: Updating checks
63  7b09def 2016—12—17T23:03:14—05:00 Ishaan: Testing bar chart plotting, will clean up
        later
64  20adaca 2016—12—17T20:40:13—05:00 oracleofnj: Some bugfixes
65  ad69dcf 2016—12—17T20:12:33—05:00 oracleofnj: Fixed extend side
66  8f76e59 2016—12—17T20:10:06—05:00 Kevin: Fixed highest_tsp to take in any number of
        players
67  0707084 2016—12—17T19:53:58—05:00 oracleofnj: Isolating
68  4479213 2016—12—17T19:47:38—05:00 oracleofnj: much longer
69  4be857f 2016—12—17T19:41:59—05:00 oracleofnj: seg fault
70  74358c1 2016—12—17T19:34:18—05:00 Kevin: Interesting program in Extend
71  ab1e1d2 2016—12—17T16:12:00—05:00 oracleofnj: Add some more stdlib funcs
72  b53463d 2016—12—17T14:51:24—05:00 GitHub: Merge pull request #125 from ExtendLang/
        stdlib—string
73  39046bc 2016—12—17T14:39:54—05:00 oracleofnj: Merge branch 'master' into stdlib—string
74  a01cc84 2016—12—17T14:39:40—05:00 oracleofnj: Use toString in toLiteral
75  ec7f10d 2016—12—17T14:38:30—05:00 GitHub: Merge pull request #102 from ExtendLang/
        circular—hotfix
76  73c454b 2016—12—17T14:34:42—05:00 GitHub: Merge branch 'master' into circular—hotfix
77  8126e2e 2016—12—17T14:24:22—05:00 oracleofnj: native toString
78  037728d 2016—12—17T13:46:34—05:00 Nigel Schuster: A lot of wrong paths make it work
79  0a4fd9d 2016—12—17T13:26:34—05:00 Nigel Schuster: Next attempt
80  56905f8 2016—12—17T13:19:59—05:00 Nigel Schuster: Merge branch 'plotting' of https://
```

```
          github.com/ExtendLang/Extend into plotting
 81  fbf3a1e 2016—12—17T13:19:52—05:00 Nigel Schuster: Manual install (maybe?)
 82  1171b71 2016—12—17T13:10:19—05:00 GitHub: Merge branch 'master' into plotting
 83  0dbf85d 2016—12—17T13:07:56—05:00 Nigel Schuster: Added libgd for travis
 84  060ae45 2016—12—17T13:01:35—05:00 oracleofnj: Merge branch 'master' into stdlib—string
 85  4402208 2016—12—17T13:01:24—05:00 oracleofnj: Add round
 86  23c2ae6 2016—12—17T13:00:05—05:00 GitHub: Merge pull request #123 from ExtendLang/size
          —asserts
 87  0ef936b 2016—12—17T12:33:31—05:00 oracleofnj: Fix merge conflicts
 88  4c51203 2016—12—17T11:59:30—05:00 oracleofnj: Right confusion
 89  c05cf61 2016—12—17T11:52:34—05:00 oracleofnj: Fix import dir bug
 90  39edbb4 2016—12—17T11:46:06—05:00 oracleofnj: Merge branch 'master' into size—asserts
 91  339cb1f 2016—12—17T11:45:54—05:00 oracleofnj: Fix merge conflict
 92  7462381 2016—12—17T11:44:50—05:00 GitHub: Merge pull request #122 from ExtendLang/
          split—stdlib
 93  61ac8f2 2016—12—17T11:38:19—05:00 oracleofnj: Size asserts
 94  606af9f 2016—12—17T11:22:36—05:00 oracleofnj: Transform asserts into more useful form;
           add calc of assert value to codegen
 95  8743e4c 2016—12—17T11:07:31—05:00 Nigel Schuster: Explicit maxfloat
 96  0f96e70 2016—12—17T11:02:09—05:00 Nigel Schuster: merge master; Keep tc around for
          testing
 97  1882524 2016—12—17T10:54:10—05:00 Nigel Schuster: Creating archive
 98  ee4f369 2016—12—17T10:40:17—05:00 oracleofnj: Combine asserts into a single expression
 99  0f0f1c8 2016—12—17T10:38:56—05:00 Nigel Schuster: Added right and left to stdlib
100  4dc1597 2016—12—17T10:35:11—05:00 Nigel Schuster: Made compiling workable
101  fa43425 2016—12—17T10:30:23—05:00 oracleofnj: Split stdlib
102  824c53c 2016—12—17T10:11:13—05:00 Nigel Schuster: Added toUpper and toLower
103  ec24177 2016—12—17T10:02:30—05:00 Nigel Schuster: Implemented to and from ASCII
104  ab2e8f8 2016—12—17T09:15:39—05:00 GitHub: Merge pull request #116 from ExtendLang/line
          —plus
105  5d1610b 2016—12—17T09:08:57—05:00 GitHub: Merge branch 'master' into line—plus
106  df3a827 2016—12—17T09:08:48—05:00 GitHub: Merge pull request #117 from ExtendLang/cmd—
          args
107  32a3487 2016—12—17T09:02:27—05:00 GitHub: Merge branch 'master' into cmd—args
108  a8f9d33 2016—12—17T09:00:23—05:00 Nigel Schuster: Args
109  bfccf0c 2016—12—17T08:58:08—05:00 Nigel Schuster: Cut down line count for plus
110  a6bc89a 2016—12—17T08:48:31—05:00 GitHub: Merge pull request #114 from ExtendLang/only
          —new—string
111  5c96b7f 2016—12—17T08:03:27—05:00 GitHub: Merge pull request #109 from ExtendLang/unop
          —bitnot
112  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
113  375bea7 2016—12—16T23:56:35—05:00 oracleofnj: Merge branch 'unop—bitnot' into remove—
          interpreter
114  fb1bd77 2016—12—16T23:54:43—05:00 oracleofnj: Clean up; remove interpreter; change
          DimInt to DimOneByOne
115  539dd75 2016—12—16T23:46:35—05:00 GitHub: Merge branch 'master' into unop—bitnot
116  5668e53 2016—12—16T23:43:57—05:00 Nigel Schuster: Using lrint instead of fptosi
117  45691eb 2016—12—16T23:35:38—05:00 GitHub: Merge pull request #111 from ExtendLang/
          global—semant
118  2cdfb8b 2016—12—16T23:33:26—05:00 GitHub: Merge branch 'master' into global—semant
119  c9500d9 2016—12—16T23:33:14—05:00 GitHub: Merge pull request #112 from ExtendLang/
          remove—function—signatures
120  0c24f54 2016—12—16T23:25:23—05:00 oracleofnj: Remove return signature from grammar and
           all test cases
121  e7f2864 2016—12—16T23:03:53—05:00 oracleofnj: Merge branch 'cleanup—1' into global—
```

```
         semant
122  567507e 2016—12—16T22:53:20—05:00 oracleofnj: Check globals; use same symbol_table
         function for semant and codegen
123  33e3942 2016—12—16T22:11:13—05:00 GitHub: Merge branch 'master' into plotting
124  55d8185 2016—12—16T22:00:30—05:00 Nigel Schuster: Removed comments and unneccessary
         files
125  629042f 2016—12—16T21:37:07—05:00 GitHub: Merge branch 'master' into unop—bitnot
126  48b139a 2016—12—16T21:34:09—05:00 Nigel Schuster: Implemented unary bitnot
127  39b02cd 2016—12—16T21:27:22—05:00 oracleofnj: Merge branch 'master' into global—semant
128  28c0983 2016—12—16T21:27:05—05:00 oracleofnj: Remove leftover printf
129  dc182df 2016—12—16T21:09:00—05:00 GitHub: Merge pull request #105 from ExtendLang/rg—
         eq
130  8cdf5c4 2016—12—16T19:31:26—05:00 oracleofnj: Expand test cases for range equality
131  41a3ccc 2016—12—16T19:18:44—05:00 GitHub: Merge branch 'master' into rg—eq
132  8dbebc1 2016—12—16T19:18:15—05:00 GitHub: Merge pull request #104 from ExtendLang/
         prevent—overlapping—formulas
133  c1431b5 2016—12—16T18:55:07—05:00 Nigel Schuster: Implemented basic subrange
         comparison
134  546536e 2016—12—16T18:47:12—05:00 oracleofnj: Detect overlapping formulas and give
         runtime error if present
135  3562e1b 2016—12—16T18:45:12—05:00 oracleofnj: Merge branch 'sr—val—fix' into prevent—
         overlapping—formulas
136  8713fa0 2016—12—16T18:42:40—05:00 oracleofnj: Checking
137  77d80b9 2016—12—16T18:26:31—05:00 Nigel Schuster: Fixed check for subrange
138  69fb0d2 2016—12—16T17:46:48—05:00 oracleofnj: Circular hotfix
139  4a3ec8d 2016—12—16T17:21:18—05:00 oracleofnj: Add concat
140  962c744 2016—12—16T12:09:00—05:00 GitHub: Merge pull request #101 from ExtendLang/
         finishing—these—range—literals
141  f234e00 2016—12—16T00:21:06—05:00 oracleofnj: Merge branch 'more—stdlib—functions'
         into finishing—these—range—literals
142  c9246ce 2016—12—16T00:20:59—05:00 oracleofnj: testing testing
143  6914039 2016—12—16T00:14:09—05:00 oracleofnj: Third time's the charm
144  4617e44 2016—12—16T00:01:12—05:00 oracleofnj: It compiles now
145  1d8e290 2016—12—15T23:42:43—05:00 oracleofnj: Fingers crossed
146  c9d28d3 2016—12—15T21:50:01—05:00 oracleofnj: Move all initializations into their own
         function; only box strings once
147  1cfdd16 2016—12—15T18:47:30—05:00 oracleofnj: Merge branch 'master' into more—stdlib—
         functions
148  19c2beb 2016—12—15T18:40:12—05:00 oracleofnj: Try a couple more things out
149  845cb04 2016—12—15T18:33:07—05:00 GitHub: Merge pull request #96 from ExtendLang/
         ternary—fix
150  4bfb3bc 2016—12—15T18:23:00—05:00 oracleofnj: Merge branch 'ternary—fix' into more—
         stdlib—functions
151  ae55ca4 2016—12—15T18:21:58—05:00 oracleofnj: Define cell_row, cell_col
152  30a5db6 2016—12—15T18:19:56—05:00 oracleofnj: Merge branch 'ternary—fix' into more—
         stdlib—functions
153  b9f1f10 2016—12—15T18:17:53—05:00 oracleofnj: What is truth?
154  ac84c2f 2016—12—15T18:15:37—05:00 oracleofnj: Fix ternary to work properly with ranges
155  1f57d91 2016—12—15T17:03:26—05:00 oracleofnj: Look at this one
156  437ba46 2016—12—15T16:56:04—05:00 oracleofnj: Try this one
157  f0edf5b 2016—12—15T16:46:52—05:00 oracleofnj: Fixing bug
158  5ba31e6 2016—12—15T14:17:52—05:00 GitHub: Merge pull request #94 from ExtendLang/nan—
         inf
159  67c5739 2016—12—15T14:17:46—05:00 GitHub: Merge pull request #93 from ExtendLang/type—
         typeof
160  48a3d5c 2016—12—15T14:05:37—05:00 oracleofnj: Improve test case
```

```
161  8f08227 2016—12—15T13:58:46—05:00 oracleofnj: Add isNaN and isInfinite to stdlib
162  cbeec74 2016—12—15T13:30:31—05:00 oracleofnj: Rename token
163  9582228 2016—12—15T13:18:09—05:00 oracleofnj: Rename type to typeof
164  d1422c7 2016—12—15T10:42:19—05:00 GitHub: Merge pull request #92 from ExtendLang/
         compiler
165  66689bb 2016—12—15T09:08:56—05:00 Nigel Schuster: added working directory option,
         doing testing completely in tmp
166  a13ae93 2016—12—15T09:08:31—05:00 GitHub: Merge pull request #91 from ExtendLang/
         sizeof
167  a31add9 2016—12—15T09:08:13—05:00 GitHub: Merge pull request #90 from ExtendLang/
         subselect—C—side
168  2e67e06 2016—12—15T09:01:06—05:00 Nigel Schuster: Added option to specify compiler,
         using clang
169  c171450 2016—12—15T02:33:48—05:00 oracleofnj: SizeOf
170  c168044 2016—12—15T00:48:35—05:00 oracleofnj: Add row(), column() to codegen, add
         print_endline() to stdlib.xtnd
171  bf9426d 2016—12—15T00:27:13—05:00 oracleofnj: Print subrange
172  407ce41 2016—12—14T23:02:02—05:00 oracleofnj: Merge in subrange_string
173  756ea8e 2016—12—14T22:51:00—05:00 oracleofnj: Ranges
174  27a8e79 2016—12—14T22:16:13—05:00 oracleofnj: Resolve RHS slice
175  876d056 2016—12—14T22:02:56—05:00 oracleofnj: Resolve RHS index
176  b59e022 2016—12—14T21:46:00—05:00 Nigel Schuster: Added method to print subragne as
         string
177  a7d53a8 2016—12—14T19:55:38—05:00 oracleofnj: Merge branch 'master' into subselect—C—
         side
178  362e85b 2016—12—14T19:55:23—05:00 GitHub: Merge pull request #88 from ExtendLang/
         subselect
179  4912fa3 2016—12—14T19:40:10—05:00 oracleofnj: Add debug print info for slice
         structures
180  c1b33f4 2016—12—14T18:58:45—05:00 oracleofnj: Builder to end all builders
181  5d400c2 2016—12—14T18:55:06—05:00 oracleofnj: Add selection builders
182  29f6e28 2016—12—14T18:20:51—05:00 oracleofnj: Make additional infix operator for
         populating structure element
183  046d096 2016—12—14T17:49:19—05:00 oracleofnj: Set up RHS slice types
184  0d20933 2016—12—14T17:28:38—05:00 GitHub: Merge branch 'master' into plotting
185  614d84f 2016—12—14T17:25:20—05:00 Nigel Schuster: Dummy commit for travis
186  0e78574 2016—12—14T17:24:04—05:00 Nigel Schuster: Merge branch 'plotting' of https://
         github.com/ExtendLang/Extend into plotting
187  2da0d7d 2016—12—14T17:23:56—05:00 Nigel Schuster: Spelling fix
188  b25c2f5 2016—12—14T16:49:17—05:00 GitHub: Merge pull request #87 from ExtendLang/make—
         a—selection
189  7a12082 2016—12—14T16:43:38—05:00 oracleofnj: Move selection test cases back into
         inputs
190  e2c08d5 2016—12—14T16:31:00—05:00 oracleofnj: Make IDs work with deref_subrange
191  02f2f0c 2016—12—14T15:21:31—05:00 GitHub: Merge pull request #86 from ExtendLang/
         include—stdlib
192  8b0503f 2016—12—14T15:18:14—05:00 GitHub: Merge branch 'master' into include—stdlib
193  1f034a0 2016—12—14T15:17:52—05:00 GitHub: Merge pull request #84 from ExtendLang/math—
         linker
194  1e6dd91 2016—12—14T14:58:44—05:00 oracleofnj: Add expected output for slurp
195  ff1a5e3 2016—12—14T14:53:38—05:00 oracleofnj: Remove extend_ prefix from all sample
         code
196  81a2828 2016—12—14T14:48:38—05:00 oracleofnj: Automatically add extend_ prefix to
         external functions
197  dcc1ed3 2016—12—14T14:30:52—05:00 oracleofnj: Fix samples
198  9b2c28f 2016—12—14T12:39:45—05:00 oracleofnj: Include stdlib automatically
```

```
199  13650ce 2016—12—14T12:35:21—05:00 Nigel Schuster: Merge branch 'math—linker' of https
         ://github.com/ExtendLang/Extend into math—linker
200  2e0d90d 2016—12—14T12:35:06—05:00 Nigel Schuster: Merge branch 'math—linker' of https
         ://github.com/ExtendLang/Extend into math—linker
201  83c689e 2016—12—14T12:34:14—05:00 Nigel Schuster: Merge branch 'math—linker' of https
         ://github.com/ExtendLang/Extend into math—linker
202  127f600 2016—12—14T12:34:07—05:00 Nigel Schuster: Include sys/resources
203  b34d97a 2016—12—14T12:03:44—05:00 GitHub: Merge branch 'master' into math—linker
204  8297f33 2016—12—14T12:01:47—05:00 GitHub: Merge pull request #85 from ExtendLang/put—
         lt—back
205  6b0c74f 2016—12—14T11:33:45—05:00 Nigel Schuster: Include sys/resources
206  37470e9 2016—12—14T11:14:06—05:00 oracleofnj: Put back LT, comment out sys/time.h
207  6bde590 2016—12—14T11:12:16—05:00 Nigel Schuster: Increasing stack size
208  6acc621 2016—12—14T11:03:31—05:00 Nigel Schuster: Disabled linking math when creating
         an intermediate
209  d87b73c 2016—12—14T10:51:58—05:00 GitHub: Merge pull request #82 from ExtendLang/hard—
         to—repro—bug
210  d126e3c 2016—12—14T00:51:00—05:00 oracleofnj: Try with time.h instead of sys/time.h
211  a535612 2016—12—14T00:48:35—05:00 oracleofnj: Remove lrints
212  e844853 2016—12—14T00:34:37—05:00 oracleofnj: Initialize all variables and remove
         pointer math; bug appears fixed
213  4c1a421 2016—12—13T22:55:07—05:00 oracleofnj: Some formula is weird
214  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
215  879eaf3 2016—12—13T22:43:17—05:00 oracleofnj: Testing
216  37f5ce2 2016—12—13T22:42:40—05:00 GitHub: Merge pull request #83 from ExtendLang/
         rounding—for—read
217  a1cfc5a 2016—12—13T22:34:21—05:00 Nigel Schuster: Added rounding at several places
218  e20f7e4 2016—12—13T21:36:13—05:00 oracleofnj: Half the time it works
219  9f97b1a 2016—12—13T20:38:08—05:00 GitHub: Merge branch 'master' into plotting
220  61bc9b6 2016—12—13T20:33:27—05:00 GitHub: Merge pull request #81 from ExtendLang/fix—
         em—all
221  4a810df 2016—12—13T19:34:29—05:00 Nigel Schuster: Corrected testcase outputs
222  ae5b8a8 2016—12—13T19:08:43—05:00 GitHub: Merge pull request #80 from ExtendLang/
         select
223  70b2704 2016—12—13T19:02:32—05:00 oracleofnj: No C99
224  15fd762 2016—12—13T18:42:21—05:00 oracleofnj: Merge branch 'master' into select
225  8e6e9ba 2016—12—13T18:42:05—05:00 GitHub: Merge pull request #78 from ExtendLang/unop—
         unary—minus
226  7a93885 2016—12—13T18:41:49—05:00 oracleofnj: Calculate all formula indices
227  07e63dc 2016—12—13T18:19:58—05:00 oracleofnj: Properly build instantiate var
228  1a29129 2016—12—13T17:24:16—05:00 oracleofnj: Replace bools with chars for
         compatibility between C and LLVM
229  12e78a3 2016—12—13T17:17:54—05:00 oracleofnj: Added debug output
230  a483282 2016—12—13T16:13:30—05:00 oracleofnj: Merge branch 'master' into unop—unary—
         minus
231  f8c9b43 2016—12—13T16:13:09—05:00 oracleofnj: Make TypeOf work
232  8146d04 2016—12—13T16:12:17—05:00 GitHub: Merge pull request #75 from ExtendLang/fix—
         more—tc
233  94afc93 2016—12—13T16:02:35—05:00 Nigel Schuster: Corrected expected TC
234  f6f8276 2016—12—13T16:00:59—05:00 Nigel Schuster: Fixed string.xtnd file
235  dcd5766 2016—12—13T15:44:38—05:00 GitHub: Merge pull request #74 from ExtendLang/fix—
         tc
236  bfe1c07 2016—12—13T15:39:45—05:00 oracleofnj: Merge branch 'master' into unop—unary—
         minus
237  d9abfc0 2016—12—13T15:38:38—05:00 GitHub: Merge branch 'master' into fix—tc
```

```
238  50ed49c 2016—12—13T15:38:04—05:00 oracleofnj: Merging in main
239  23328f1 2016—12—13T15:37:18—05:00 GitHub: Merge pull request #73 from ExtendLang/and—
         or—xor
240  324779a 2016—12—13T15:32:26—05:00 Nigel Schuster: Corrected expected value
241  fafe2e6 2016—12—13T15:29:21—05:00 Nigel Schuster: Fixed string tc
242  022f05c 2016—12—13T15:23:59—05:00 Nigel Schuster: Fixed testcase
243  b12fe37 2016—12—13T15:18:57—05:00 Nigel Schuster: Implemented and, or and xor
244  90cbaa0 2016—12—13T15:16:31—05:00 Nigel Schuster: Added left and right shift
245  571ee7e 2016—12—13T14:56:05—05:00 Nigel Schuster: Merge branch 'power' of https://
         github.com/ExtendLang/Extend into power
246  aeab40d 2016—12—13T14:55:57—05:00 Nigel Schuster: Removed unneccessary level of
         indirection
247  e377567 2016—12—13T14:53:28—05:00 GitHub: Merge branch 'master' into power
248  6ad8512 2016—12—13T14:53:11—05:00 GitHub: Merge pull request #69 from ExtendLang/unop—
         unary—minus
249  71f395d 2016—12—13T14:46:27—05:00 Nigel Schuster: Power to the people of Extend
250  6a04209 2016—12—13T14:45:46—05:00 oracleofnj: Fix merge conflict
251  edb0ecc 2016—12—13T14:43:32—05:00 oracleofnj: Add unary minus
252  668a0eb 2016—12—13T14:37:19—05:00 GitHub: Merge pull request #68 from ExtendLang/mod—
         div
253  866b68f 2016—12—13T14:32:18—05:00 Nigel Schuster: Added modulo and division operation
254  46d5aa6 2016—12—13T14:26:35—05:00 oracleofnj: Merge branch 'master' into unop—typeof
255  84dfc33 2016—12—13T14:26:25—05:00 Nigel Schuster: Crunched some code
256  76210eb 2016—12—13T14:26:18—05:00 oracleofnj: Start on it
257  f4d5a81 2016—12—13T14:22:12—05:00 Nigel Schuster: Merge branch 'master' into
         simplification
258  f873242 2016—12—13T14:21:26—05:00 GitHub: Merge pull request #65 from ExtendLang/
         subtraction
259  fc94112 2016—12—13T14:20:35—05:00 Nigel Schuster: Added multiplication
260  6c26c2c 2016—12—13T14:19:07—05:00 GitHub: Merge branch 'master' into subtraction
261  4afd78e 2016—12—13T14:18:55—05:00 GitHub: Merge pull request #64 from ExtendLang/
         refactor—boolean—binops
262  d4d4388 2016—12—13T14:15:58—05:00 GitHub: Merge branch 'master' into refactor—boolean—
         binops
263  bd90241 2016—12—13T14:14:17—05:00 GitHub: Merge branch 'master' into subtraction
264  4042259 2016—12—13T14:13:09—05:00 Nigel Schuster: Added subtraction
265  663f399 2016—12—13T14:12:57—05:00 oracleofnj: Remove wildcard from BinOp pattern match
266  82a3db2 2016—12—13T14:11:31—05:00 Nigel Schuster: Merge branch 'master' into
         subtraction
267  1bf6bed 2016—12—13T14:09:47—05:00 oracleofnj: Add TransformedAway exception for LogAnd
          and LogOr
268  c7d4162 2016—12—13T14:02:13—05:00 GitHub: Merge pull request #63 from ExtendLang/more—
         binops
269  952778e 2016—12—13T14:01:54—05:00 oracleofnj: Change Lt, Lte in grammar; implement GTE
270  97821c8 2016—12—13T13:47:52—05:00 oracleofnj: GT
271  1e1f973 2016—12—13T13:44:36—05:00 Nigel Schuster: Subtraction
272  e0a883a 2016—12—13T13:37:57—05:00 oracleofnj: Remove NotEq from AST since != is parsed
          to UnOp(LogNot,BinOp(Eq,...))
273  cc40008 2016—12—13T12:49:33—05:00 GitHub: Merge pull request #60 from ExtendLang/
         addition2
274  7123ebc 2016—12—13T12:41:09—05:00 GitHub: Merge branch 'master' into addition2
275  a656f57 2016—12—13T12:38:12—05:00 GitHub: Merge pull request #61 from ExtendLang/debug
         —unop
276  c3a96a9 2016—12—13T12:37:31—05:00 Nigel Schuster: Merge branch 'master' into plotting
277  f59d962 2016—12—13T12:34:49—05:00 Nigel Schuster: Moved make of lib to travis script
278  eb134b3 2016—12—13T12:29:53—05:00 Nigel Schuster: Moved testcases
```

```
279  044c6bd 2016—12—13T12:29:07—05:00 Nigel Schuster: Fixed off by one error
280  a64cc15 2016—12—13T12:14:45—05:00 oracleofnj: Add Debug expr
281  59858a0 2016—12—13T11:33:12—05:00 oracleofnj: Whoops no space
282  0426f34 2016—12—13T11:30:26—05:00 oracleofnj: Add test case
283  49ffa86 2016—12—13T11:19:14—05:00 GitHub: Merge branch 'master' into addition2
284  81533f4 2016—12—13T11:13:44—05:00 GitHub: Merge pull request #59 from ExtendLang/equal
        —rights
285  3cdaa5a 2016—12—13T11:12:41—05:00 Nigel Schuster: String addition
286  64d1760 2016—12—13T11:04:55—05:00 oracleofnj: Wake up please, GitHub
287  840aeaf 2016—12—13T10:48:03—05:00 oracleofnj: Remove usage demonstration
288  61ff439 2016—12—13T03:26:35—05:00 oracleofnj: Add string equality and test cases
289  f3112e9 2016—12—13T01:57:10—05:00 oracleofnj: Reduce cut & paste
290  08ce677 2016—12—13T01:35:46—05:00 oracleofnj: Remove obsolete testing file
291  ae8a07e 2016—12—13T01:23:26—05:00 oracleofnj: Merge branch 'print_value_p' into equal—
        rights
292  6090713 2016—12—13T01:22:47—05:00 oracleofnj: Use correct printf specifier
293  862b38c 2016—12—13T01:19:14—05:00 oracleofnj: Merge branch 'print_value_p' into equal—
        rights
294  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)
295  50281b1 2016—12—13T00:47:28—05:00 oracleofnj: Numeric equality
296  0f76aa4 2016—12—12T22:30:15—05:00 oracleofnj: Remove print flags
297  200b8b6 2016—12—12T22:16:15—05:00 GitHub: Merge pull request #57 from ExtendLang/
        addition2
298  da7c543 2016—12—12T12:43:31—05:00 Nigel Schuster: Setting flag for addition
299  7e7276b 2016—12—12T12:37:35—05:00 Nigel Schuster: Merge branch 'master' into addition2
300  8834635 2016—12—12T10:18:51—05:00 GitHub: Merge pull request #55 from ExtendLang/
        runtime
301  53ae9e0 2016—12—12T10:06:24—05:00 GitHub: Merge branch 'master' into runtime
302  6ed303e 2016—12—12T09:43:57—05:00 GitHub: Merge pull request #56 from ExtendLang/
        truthy—fix
303  ae49ce6 2016—12—12T01:15:29—05:00 oracleofnj: Remove extra file
304  7fe6a22 2016—12—12T01:11:53—05:00 oracleofnj: Falsey fix
305  d1e196d 2016—12—12T00:23:13—05:00 Nigel Schuster: Extracted runtime into seperate file
306  ecc620e 2016—12—12T00:17:06—05:00 GitHub: Merge pull request #54 from ExtendLang/final
        —draft—for—real
307  4c8caa5 2016—12—12T00:09:16—05:00 GitHub: Merge branch 'master' into final—draft—for—
        real
308  04d3b57 2016—12—12T00:00:29—05:00 GitHub: Merge pull request #39 from ExtendLang/more—
        lrm—ed
309  39025b0 2016—12—11T23:59:18—05:00 Nigel Schuster: Fixed examples, made small
        corrections
310  a875b41 2016—12—11T23:51:30—05:00 GitHub: Merge pull request #53 from ExtendLang/
        truthy
311  616dd34 2016—12—11T23:15:54—05:00 oracleofnj: Merge branch 'master' into truthy
312  0fa8255 2016—12—11T23:14:42—05:00 oracleofnj: Apparently still needs some work
313  78584d7 2016—12—11T23:09:07—05:00 oracleofnj: Thanks a lot Travis
314  b5673d2 2016—12—11T22:51:52—05:00 oracleofnj: TERRRRRRRRR NARRRRRRR
        EEEEEEEEEEEEEEEEEEE
315  b81bc1b 2016—12—11T22:04:25—05:00 oracleofnj: Maybe Truthy
316  b95d14f 2016—12—11T21:02:28—05:00 GitHub: Merge pull request #50 from ExtendLang/
        builder—hotfix
317  6dea96f 2016—12—11T20:40:47—05:00 oracleofnj: So many builders
318  8aa125f 2016—12—11T20:15:52—05:00 Nigel Schuster: Made som rpgroess
319  2a905c7 2016—12—11T19:15:47—05:00 GitHub: Merge pull request #47 from ExtendLang/
```

```
          function-parameter
320  2bc6c85 2016-12-11T19:11:33-05:00 oracleofnj: Add combined test case
321  860a11b 2016-12-11T19:04:35-05:00 oracleofnj: Merge branch 'master' into function-
          parameter
322  8c3499e 2016-12-11T19:03:39-05:00 oracleofnj: Remove extraneous printlines
323  99418c0 2016-12-11T19:02:31-05:00 oracleofnj: Make function parameters work
324  6c00a72 2016-12-11T18:45:46-05:00 Nigel Schuster: Some progress
325  387559b 2016-12-11T18:39:00-05:00 oracleofnj: First attempt
326  18fc1be 2016-12-11T18:08:11-05:00 GitHub: Merge pull request #45 from ExtendLang/empty
327  d7590da 2016-12-11T17:42:46-05:00 GitHub: Merge branch 'master' into plotting
328  f7e9be8 2016-12-11T16:30:05-05:00 GitHub: Merge branch 'master' into empty
329  f1dd8a5 2016-12-11T16:18:44-05:00 GitHub: Merge pull request #46 from ExtendLang/
          actually-make-global-scope
330  50366f4 2016-12-11T15:38:05-05:00 oracleofnj: Make sure locals are properly masking
          globals
331  046c7cc 2016-12-11T15:30:53-05:00 oracleofnj: Make globals work, fix bug
332  a844a46 2016-12-11T15:14:09-05:00 oracleofnj: So close
333  18db166 2016-12-11T15:05:42-05:00 GitHub: Merge branch 'master' into empty
334  67849f0 2016-12-11T15:01:52-05:00 oracleofnj: Make the global scope object
335  393d02c 2016-12-11T14:25:02-05:00 Nigel Schuster: Implemented empty, small flag
          setting fix
336  3c4681d 2016-12-11T13:31:12-05:00 GitHub: Merge pull request #44 from ExtendLang/float
          -display-hotfix
337  7be1001 2016-12-11T13:26:55-05:00 GitHub: Merge branch 'master' into float-display-
          hotfix
338  b192a23 2016-12-11T13:26:48-05:00 Nigel Schuster: Added gdchart compile step
339  abcffd0 2016-12-11T13:19:05-05:00 GitHub: Merge pull request #42 from ExtendLang/
          encapsulate-build-scope
340  556da44 2016-12-11T13:18:15-05:00 oracleofnj: Floating point math hotfix
341  0ad195e 2016-12-11T12:42:42-05:00 oracleofnj: Merge branch 'master' into encapsulate-
          build-scope
342  9caf464 2016-12-11T12:41:40-05:00 oracleofnj: Encapsulate a little more of building
          the scope
343  1ae8d43 2016-12-11T12:23:04-05:00 Ishaan: Add new gitignore
344  6278c7b 2016-12-11T12:18:49-05:00 Ishaan: Rebase and add gdchart in lib/
345  5594687 2016-12-11T12:13:20-05:00 Ishaan: Remove images from version control
346  294a6db 2016-12-11T12:13:20-05:00 Ishaan: Write to file instead of stdout
347  08e9f75 2016-12-11T12:11:13-05:00 Ishaan: Add harcoded graph functionality
348  d65aad4 2016-12-11T12:09:28-05:00 GitHub: Merge pull request #40 from ExtendLang/make-
          global-scope
349  b5b33f1 2016-12-11T12:09:12-05:00 Ishaan: Update gitignore to avoid the gdchart
          package
350  6746e8a 2016-12-11T12:09:12-05:00 Ishaan: Checking gif
351  83c2e09 2016-12-11T12:09:12-05:00 Ishaan: Add hardcoded plot function without params
          or installation
352  0f5a6ba 2016-12-11T12:04:05-05:00 oracleofnj: Merge branch 'master' into make-global-
          scope
353  56b58d9 2016-12-11T12:01:28-05:00 oracleofnj: Encapsulate build_var_defns
354  f25e5b3 2016-12-11T11:43:19-05:00 oracleofnj: Only construct var_defns once
355  9cee2fc 2016-12-11T10:07:36-05:00 Nigel Schuster: Testcases (#38)
356  f3f4bef 2016-12-11T00:45:44-05:00 oracleofnj: Make global variable to hold vardefns
357  a0ed757 2016-12-10T23:31:38-05:00 Nigel Schuster: Edited explanation for row() and
          column()
358  7c50ef2 2016-12-10T23:27:07-05:00 Nigel Schuster: Added info for strings
359  738e41b 2016-12-10T23:24:20-05:00 Nigel Schuster: Added boolean example
360  5377fdf 2016-12-10T23:19:26-05:00 Nigel Schuster: Added arithmetic example
```

361 a8f4ad9 2016—12—10T21:28:18—05:00 oracleofnj: Isolate the part of building a scope for reuse with global variables
362 58f7a4d 2016—12—10T18:05:01—05:00 Nigel Schuster: Performing copy before returning, so that memory can be freed with alloca
363 c0e56aa 2016—12—10T17:07:00—05:00 GitHub: Merge pull request #37 from ExtendLang/dereference
364 a4b35df 2016—12—10T16:42:17—05:00 Nigel Schuster: Removed obsolete methods
365 cf08a8c 2016—12—10T16:36:20—05:00 GitHub: Merge branch 'master' into dereference
366 ef0e5e7 2016—12—10T16:36:03—05:00 GitHub: Merge pull request #36 from ExtendLang/comp—warn
367 0177dc2 2016—12—10T16:35:50—05:00 GitHub: Merge pull request #35 from ExtendLang/linker
368 127f99d 2016—12—10T16:35:41—05:00 GitHub: Merge pull request #34 from ExtendLang/rel—import
369 b2e881d 2016—12—10T16:35:31—05:00 GitHub: Merge pull request #33 from ExtendLang/ts—fix
370 ce833d4 2016—12—10T16:14:34—05:00 Nigel Schuster: Dereferencing 1x1 subrange
371 e259556 2016—12—10T13:53:12—05:00 Nigel Schuster: Removed nodefaultlibs directive
372 09c3961 2016—12—10T13:50:19—05:00 Nigel Schuster: Modified linker to work for travis
373 36d662a 2016—12—10T13:37:27—05:00 Nigel Schuster: Attempt to link math
374 2d4564a 2016—12—10T13:22:14—05:00 Nigel Schuster: Linking math library
375 38ba6e6 2016—12—10T13:18:39—05:00 Nigel Schuster: Suppressing compiler warnings
376 9deac9b 2016—12—10T13:06:39—05:00 Nigel Schuster: Modified compile script. Removed debug output
377 d35607b 2016—12—10T13:04:30—05:00 Nigel Schuster: Simpler testscript
378 d37dac2 2016—12—10T12:36:45—05:00 Nigel Schuster: Fixed duplicate import issue
379 31c26bc 2016—12—10T12:30:29—05:00 Nigel Schuster: Added cmd args to link file
380 a350720 2016—12—10T11:40:50—05:00 Nigel Schuster: Switched import style from root directory to relative path
381 90e39b0 2016—12—10T11:24:19—05:00 Nigel Schuster: Fixed issue in testscript that might report false results when it fails early
382 718ecd3 2016—12—10T03:09:18—05:00 oracleofnj: Some changes to LRM; add if(a,b,c)
383 6a8f836 2016—12—09T18:29:22—05:00 GitHub: Merge pull request #24 from ExtendLang/final—draft—lrm
384 fc886a9 2016—12—09T18:23:52—05:00 oracleofnj: Merge branch 'final—draft—lrm'
385 cda63cb 2016—12—09T18:23:24—05:00 oracleofnj: Fix merge conflict
386 eac9e77 2016—12—09T18:04:08—05:00 GitHub: Merge pull request #29 from ExtendLang/refactor
387 fe825f4 2016—12—09T17:55:39—05:00 oracleofnj: Compact last bit
388 b02dbbe 2016—12—09T17:49:00—05:00 oracleofnj: Give formula functions names
389 edd7aa4 2016—12—09T17:40:57—05:00 Nigel Schuster: Removed artifcats
390 9b49e20 2016—12—09T17:37:59—05:00 Nigel Schuster: Fixed I/O testcases
391 a4ad4b1 2016—12—09T17:18:13—05:00 Nigel Schuster: Merge
392 b07398b 2016—12—09T17:17:19—05:00 Nigel Schuster: Added macro for function definition
393 ed01567 2016—12—09T17:17:06—05:00 oracleofnj: Make sizeof not break tests
394 a0a7054 2016—12—09T17:01:20—05:00 oracleofnj: Use symbol table
395 56fd61b 2016—12—09T16:11:10—05:00 oracleofnj: Merge branch 'refactor' of https://github.com/ExtendLang/Extend into refactor
396 38aedba 2016—12—09T16:10:35—05:00 oracleofnj: Create symbol table
397 dfb702e 2016—12—09T16:01:08—05:00 Nigel Schuster: Converted more to value_p from subrange_p
398 e963186 2016—12—09T15:42:35—05:00 Nigel Schuster: Made example TC work
399 eb76234 2016—12—09T11:14:58—05:00 Nigel Schuster: Made Hello World work again
400 08aeb70 2016—12—09T02:13:09—05:00 oracleofnj: Done for the night
401 cb39114 2016—12—09T01:35:36—05:00 oracleofnj: More refactoring
402 7974bbd 2016—12—08T23:53:31—05:00 oracleofnj: Banish the term extern

```
403  49af972 2016—12—08T23:45:30—05:00 oracleofnj: Add a couple comments
404  0fbf461 2016—12—08T21:52:24—05:00 oracleofnj: Get my bearings
405  5ecb599 2016—12—08T19:47:51—05:00 Nigel Schuster: Added some documentation
406  65066fc 2016—12—08T12:18:57—05:00 Nigel Schuster: Added name display for variable
407  fb18949 2016—12—07T23:44:17—05:00 oracleofnj: Merge branch 'master' into final—draft—
        lrm
408  4aab3dc 2016—12—07T23:43:25—05:00 oracleofnj: Update PDF
409  ed44d27 2016—12—07T23:43:01—05:00 oracleofnj: Fix failing test cases
410  9354fa7 2016—12—07T23:06:36—05:00 oracleofnj: Final draft candidate
411  78649f4 2016—12—07T18:09:46—05:00 oracleofnj: Almost done
412  05ded19 2016—12—07T15:47:52—05:00 oracleofnj: More work
413  f985cc8 2016—12—07T12:14:59—05:00 Nigel Schuster: Merge branch 'finish—transformations
        ' into get—val—rev
414  4b58ce9 2016—12—07T12:13:23—05:00 Nigel Schuster: Tried to add more instructions
415  0722412 2016—12—07T11:32:11—05:00 oracleofnj: Working
416  099efe7 2016—12—07T10:48:35—05:00 Nigel Schuster: Making progress on evaluating
        dimensions
417  fa09df7 2016—12—07T09:51:23—05:00 Nigel Schuster: Finally it works
418  cbb0577 2016—12—07T02:35:06—05:00 oracleofnj: Still WIP
419  e3c9436 2016—12—07T00:44:22—05:00 oracleofnj: WIP
420  b265e74 2016—12—07T00:41:23—05:00 Nigel Schuster: test commit to look at
421  18bb182 2016—12—07T00:35:06—05:00 oracleofnj: Still work in progress
422  a4554c0 2016—12—06T23:14:32—05:00 Nigel Schuster: At least it compiles
423  3432484 2016—12—06T22:42:22—05:00 Nigel Schuster: Getting closer. Need to add var_defn
         wrapper in build_formula
424  05145ca 2016—12—06T21:10:11—05:00 Nigel Schuster: Minor fix
425  af69b92 2016—12—06T17:23:45—05:00 oracleofnj: More updates
426  a65c24e 2016—12—06T16:14:10—05:00 oracleofnj: Merge branch 'master' into finish—
        transformations
427  85a4ccb 2016—12—06T16:12:31—05:00 oracleofnj: LRM update part 1
428  174a7b8 2016—12—06T11:09:31—05:00 Nigel Schuster: Made partial progress on
        implementing variable instanciation and such
429  90fc58e 2016—12—05T22:14:41—05:00 GitHub: Merge pull request #23 from ExtendLang/read—
        empty
430  767851d 2016—12—05T16:18:17—05:00 Nigel Schuster: Finished C side implementation of
        getVal
431  6b837d4 2016—12—05T16:06:34—05:00 Nigel Schuster: Merge branch 'master' into get—val
432  04c2c65 2016—12—05T15:53:35—05:00 oracleofnj: Add slurp by passing 0 max bytes
433  d8cf316 2016—12—05T14:46:46—05:00 oracleofnj: Start handling empty
434  910bd01 2016—12—05T14:27:07—05:00 GitHub: Merge pull request #21 from ExtendLang/
        fileio
435  1ce7f83 2016—12—05T14:18:41—05:00 oracleofnj: Create patch file
436  88480fb 2016—12—05T13:36:28—05:00 GitHub: Merge branch 'master' into fileio
437  29d02d9 2016—12—05T13:34:27—05:00 oracleofnj: Fix merge conflict — keep expr_loc
438  52e7a8a 2016—12—05T13:32:54—05:00 GitHub: Merge pull request #22 from ExtendLang/rm—
        micro
439  bfa906b 2016—12—05T13:28:03—05:00 oracleofnj: Fix off—by—one bug
440  eb8dd71 2016—12—05T13:20:03—05:00 oracleofnj: Address issues
441  f1b11ee 2016—12—05T12:46:35—05:00 Nigel Schuster: Skeleton for get_val
442  e4e5e26 2016—12—05T09:25:17—05:00 Nigel Schuster: Removed microc reference
        implementation
443  270da2b 2016—12—05T02:40:59—05:00 GitHub: Merge branch 'master' into fileio
444  b928e98 2016—12—05T02:40:10—05:00 Ishaan: Remove bloat
445  894b511 2016—12—05T02:32:49—05:00 Ishaan: Added testcase
446  62b8e83 2016—12—05T02:30:16—05:00 Ishaan: Added fwrite implementation
447  77a23ae 2016—12—05T01:39:30—05:00 Ishaan: Added read
```

```
448  46e9b58 2016—12—05T00:07:16—05:00 Ishaan: Make refactoring changes and new helpers
449  a5b9066 2016—12—04T14:00:30—05:00 GitHub: Merge pull request #20 from ExtendLang/lhs—
         all—ids
450  35e9471 2016—12—04T13:38:44—05:00 oracleofnj: Put back Id(s) as it was
451  641d454 2016—12—04T13:36:36—05:00 oracleofnj: Always transform to ID on LHS, even for
         LitInts
452  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
453  f47f2ba 2016—12—04T10:30:44—05:00 oracleofnj: Add error handling to close() and add a
         couple test cases
454  e95a95a 2016—12—04T10:07:01—05:00 oracleofnj: Add assertSingleNumber and get_number to
         eliminate more copy & paste
455  543e720 2016—12—04T09:47:03—05:00 oracleofnj: Add new_number() to eliminate some copy
         and paste
456  d7f10c9 2016—12—04T02:31:03—05:00 Ishaan: Tentative drafts of fileio functions
457  7d81e43 2016—12—04T00:15:20—05:00 oracleofnj: add diagnostic prinfs
458  868d9a4 2016—12—03T23:46:01—05:00 Ishaan: Cleanup
459  aa1e014 2016—12—03T23:42:46—05:00 Ishaan: Add file pointer array
460  88d05de 2016—12—03T18:38:34—05:00 Ishaan: Working on fopen
461  36f5848 2016—12—03T14:07:39—05:00 oracleofnj: Merge branch ’master’ into finish—
         transformations
462  2ae2b83 2016—12—03T14:06:40—05:00 GitHub: Merge pull request #15 from ExtendLang/
         stdlib—fun
463  7c78a23 2016—12—03T14:02:51—05:00 oracleofnj: Move test_fabs out of regression test
         suite
464  0a8055b 2016—12—03T13:48:19—05:00 oracleofnj: make test | grep REGRESSION
465  a24742b 2016—12—02T22:50:43—05:00 Kevin: Merged stdlib with master
466  5243c5a 2016—12—02T18:16:36—05:00 Kevin: Removed magic numbers and add fabs test
467  330bec3 2016—12—02T13:49:34—05:00 oracleofnj: Merge branch ’master’ into finish—
         transformations
468  8a60995 2016—12—01T23:38:54—05:00 GitHub: Merge pull request #18 from ExtendLang/
         parser—error
469  f0d33e2 2016—12—01T23:18:39—05:00 oracleofnj: Move error handling
470  3b24c3a 2016—12—01T23:16:53—05:00 oracleofnj: Adjust test script
471  60a732f 2016—12—01T22:55:28—05:00 oracleofnj: Merge branch ’master’ into parser—error
472  5dec6a2 2016—12—01T22:55:05—05:00 oracleofnj: Thank you Nigel!!!
473  96a3028 2016—12—01T22:19:21—05:00 GitHub: Merge pull request #16 from ExtendLang/fail—
         silent
474  6c3696c 2016—12—01T21:59:40—05:00 oracleofnj: Figure out why test is failing
475  7912d5a 2016—12—01T21:26:03—05:00 GitHub: Merge branch ’master’ into fail—silent
476  9702e5b 2016—12—01T21:14:35—05:00 oracleofnj: Merge branch ’master’ into finish—
         transformations
477  5bdd52c 2016—12—01T21:13:45—05:00 GitHub: Merge pull request #17 from ExtendLang/
         lexbuf—pos
478  8893255 2016—12—01T20:35:04—05:00 oracleofnj: Add a couple test cases
479  2868653 2016—12—01T20:23:01—05:00 oracleofnj: Use lexbuf.lex_curr_p to calculate
         position
480  8c7b6ce 2016—12—01T18:59:49—05:00 GitHub: Merge pull request #11 from ExtendLang/
         parse_error
481  2885ac7 2016—12—01T18:56:15—05:00 Ishaan: Added test case for string
482  047cfec 2016—12—01T18:42:04—05:00 oracleofnj: Add short circuiting test cases
483  6acd7f6 2016—12—01T18:31:33—05:00 oracleofnj: Merge remote—tracking branch ’origin/
         fail—silent’ into finish—transformations
484  72360f4 2016—12—01T17:09:08—05:00 Nigel Schuster: Minified error output for outputs
         that have not passed yet
485  5762112 2016—12—01T16:04:06—05:00 oracleofnj: Get rid of wildcard pattern match in
```

173

```
              interpreter
486 a90a343 2016—12—01T15:59:40—05:00 oracleofnj: Merge branch 'master' into finish—
              transformations
487 85bc21d 2016—12—01T15:59:05—05:00 oracleofnj: Remove unnecessary file
488 81fe565 2016—12—01T15:58:40—05:00 oracleofnj: Finish range literals
489 e9fb1c2 2016—12—01T15:04:03—05:00 Ishaan: Added increment to string buffer and tests
490 eb7c1e8 2016—12—01T15:04:03—05:00 Ishaan: Add partial character indexing
491 df09aea 2016—12—01T15:04:03—05:00 Ishaan: Add expected parse testcase intermediate
492 712a710 2016—12—01T15:04:03—05:00 Ishaan: Added tentative scanner—level line number
493 bf4ee6c 2016—12—01T15:04:03—05:00 Ishaan: Added SyntaxError Exception at scan level
494 da41520 2016—12—01T14:54:21—05:00 oracleofnj: So close
495 7abb394 2016—12—01T14:07:58—05:00 GitHub: Merge pull request #14 from ExtendLang/
              sinner
496 e0b7fdb 2016—12—01T14:05:38—05:00 Nigel Schuster: Rename empty to new_val
497 2cabadc 2016—12—01T11:58:03—05:00 oracleofnj: Merge branch 'master' into finish—
              transformations
498 6ea8cff 2016—12—01T10:10:26—05:00 Nigel Schuster: Using define instead of magic
              numbers
499 cd7d261 2016—12—01T10:07:10—05:00 Nigel Schuster: Merge branch 'master' into sinner
500 13cd317 2016—12—01T10:06:25—05:00 GitHub: Merge pull request #13 from ExtendLang/
              value_p
501 cf36f70 2016—12—01T09:47:38—05:00 oracleofnj: Sample digits function
502 4eeed07 2016—12—01T01:02:56—05:00 Ishaan: Change print return type to empty
503 fa42f27 2016—12—01T00:41:47—05:00 Kevin: Fixed acos function
504 53d34ad 2016—12—01T00:29:32—05:00 Nigel Schuster: Moved double values type to numeric
505 f769c61 2016—12—01T00:18:07—05:00 Nigel Schuster: Merge branch 'sinner' into stdlib—
              fun
506 3986f38 2016—12—01T00:17:21—05:00 Nigel Schuster: Merge branch 'value_p' into sinner
507 5bd87f9 2016—12—01T00:14:45—05:00 Nigel Schuster: Explicitly declaring to link math
              library
508 4604545 2016—12—01T00:12:08—05:00 Nigel Schuster: Consistently using floats
509 38b9824 2016—11—30T23:46:14—05:00 Nigel Schuster: Merge branch 'value_p' into sinner
510 3303575 2016—11—30T23:45:25—05:00 Nigel Schuster: Explicitly declaring to link math
              library
511 31a74ec 2016—11—30T23:35:34—05:00 Nigel Schuster: Merge branch 'master' into value_p
512 7f0bc86 2016—11—30T23:04:34—05:00 Kevin: Finished remainder of stdlib
513 cd160df 2016—11—30T22:50:18—05:00 Kevin: Added more c functions to stdlib
514 e085977 2016—11—30T19:59:57—05:00 Nigel Schuster: Made sin function work
515 206ee5a 2016—11—30T19:07:28—05:00 Nigel Schuster: Moved all function signatures to
              value_p return value
516 effc20b 2016—11—30T18:45:52—05:00 GitHub: Merge pull request #12 from ExtendLang/easy—
              compile
517 3b6d7b7 2016—11—30T17:51:19—05:00 Nigel Schuster: Added script to compile and link
518 febcff8 2016—11—30T15:54:45—05:00 oracleofnj: Add oddball formula test case and try
              out theory for range literal
519 4a1ff4f 2016—11—30T14:54:05—05:00 oracleofnj: Finish reducing Ternary to
              ReducedTernary
520 8f0a981 2016—11—30T12:35:43—05:00 oracleofnj: Working on reducing ternaries
521 d3c5812 2016—11—30T02:39:58—05:00 oracleofnj: Finish desugaring switch
522 0a22713 2016—11—30T00:09:10—05:00 oracleofnj: Getting ready to ternarize switch
523 84f016a 2016—11—29T21:54:15—05:00 oracleofnj: Fix bug in switch() with default case
524 d331b7a 2016—11—29T17:33:41—05:00 oracleofnj: Give desugaring variables easier—to—read
               names for debugging purposes
525 36f8de5 2016—11—29T16:14:46—05:00 oracleofnj: Missed one
526 d96da34 2016—11—29T16:13:21—05:00 oracleofnj: Transform &&, || into ternary
              expressions to support proper short—circuit evaluation
```

```
527  3a8efbc 2016—11—28T23:05:28—05:00 GitHub: Merge pull request #9 from ExtendLang/func—
         calls
528  7a2af49 2016—11—28T20:33:53—05:00 Nigel Schuster: Removed another ocaml 4.3 dep
529  468e79f 2016—11—28T19:50:53—05:00 Nigel Schuster: Added ocaml 4.3 as dep for travis (
         hopefully this works)
530  a408761 2016—11—28T19:35:49—05:00 Nigel Schuster: Fixed String.equal
531  90c3caf 2016—11—27T22:52:14—05:00 Nigel Schuster: Fixed interpreter for now
532  a18da78 2016—11—27T22:42:27—05:00 Nigel Schuster: Added accidentally created file
533  5647312 2016—11—27T22:41:22—05:00 Nigel Schuster: Made extern function calls work
534  872aa8c 2016—11—27T13:52:44—05:00 Nigel Schuster: Merge branch 'func—calls' of https
         ://github.com/ExtendLang/Extend into func—calls
535  26ef1cc 2016—11—27T13:51:06—05:00 Nigel Schuster: Merging list of functions
536  877336f 2016—11—27T12:15:11—05:00 GitHub: Merge branch 'master' into func—calls
537  5b3edb0 2016—11—27T12:14:43—05:00 GitHub: Merge pull request #8 from ExtendLang/stdlib
         —template
538  374273f 2016—11—27T12:13:52—05:00 Nigel Schuster: Function calls work now
539  952aab8 2016—11—27T09:54:12—05:00 Nigel Schuster: Merge extern
540  ac6268f 2016—11—26T23:06:00—05:00 Nigel Schuster: Boxing ints, added unop sizeof,
         actually returning subrange not dummy object
541  ca07be3 2016—11—26T21:27:19—05:00 Nigel Schuster: Unboxing hello world to and from
         subrange
542  aef6c19 2016—11—26T16:55:48—05:00 Nigel Schuster: Made Hello World somewhat workable
543  cfb637e 2016—11—25T18:27:37—05:00 Nigel Schuster: Fixed faulty setup on call
544  ebf926a 2016—11—25T17:48:57—05:00 Nigel Schuster: Added template in C
545  554fbb2 2016—11—23T22:28:29—05:00 oracleofnj: Better error message for WrongNumberArgs
546  f09e40e 2016—11—23T12:47:39—05:00 oracleofnj: Make sequence work
547  053980b 2016—11—22T16:02:27—05:00 oracleofnj: Actually commit all the extern stuff
548  0e0fa23 2016—11—22T14:36:54—05:00 Nigel Schuster: Added extern in Ast
549  aac63be 2016—11—21T23:52:25—05:00 oracleofnj: Better duplicate definition checking
550  08e2d07 2016—11—21T23:29:28—05:00 oracleofnj: Check assertions before evaluating fn
         return expression
551  69fa332 2016—11—21T18:01:23—05:00 oracleofnj: Add size assertions
552  22541c4 2016—11—21T12:48:34—05:00 oracleofnj: Fix bug in Call()
553  9a1d24b 2016—11—21T12:39:41—05:00 oracleofnj: Working on crazy bug
554  a485cee 2016—11—20T22:13:46—05:00 oracleofnj: Add test case for foo([m, n] arg)
555  10afe9a 2016—11—20T22:07:17—05:00 oracleofnj: Expand function signature
556  325e9ba 2016—11—20T18:53:52—05:00 oracleofnj: Well, this is awkward
557  0a76dc9 2016—11—20T18:41:12—05:00 oracleofnj: Add check of return value
558  488e34e 2016—11—20T18:31:39—05:00 oracleofnj: Add sample #1
559  93eebc5 2016—11—20T18:27:23—05:00 oracleofnj: Add semantic checking to make sure
         functions and variables on RHS exist
560  881f164 2016—11—20T17:22:40—05:00 oracleofnj: Check RHS slice to ensure end > start,
         otherwise evaluate to empty
561  442ae91 2016—11—20T11:42:54—05:00 GitHub: Merge pull request #73 from Neitsch/
         interpreter—global
562  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
563  367bc2b 2016—11—20T00:33:17—05:00 GitHub: Merge pull request #72 from Neitsch/codegen—
         part—app—fix
564  bdca834 2016—11—20T00:31:04—05:00 GitHub: Merge branch 'master' into codegen—part—app—
         fix
565  e956238 2016—11—20T00:28:49—05:00 GitHub: Merge pull request #71 from Neitsch/tc—fixes
566  9b742d1 2016—11—20T00:24:39—05:00 Nigel Schuster: Fixed partial function application
         warning
567  32f2989 2016—11—20T00:20:51—05:00 GitHub: Merge branch 'master' into tc—fixes
568  f87cb94 2016—11—20T00:20:35—05:00 GitHub: Merge pull request #69 from Neitsch/
```

```
      regression—tests
569  842ee5a 2016—11—20T00:18:56—05:00 GitHub: Merge branch 'master' into regression—tests
570  6d73717 2016—11—19T23:55:35—05:00 GitHub: Merge pull request #66 from Neitsch/fix—test
      —cases
571  05f317a 2016—11—19T22:37:36—05:00 Nigel Schuster: Fixed output on TCs
572  aa1d974 2016—11—19T22:33:40—05:00 Nigel Schuster: Fixed expected value for ternary
573  ab7653a 2016—11—19T22:32:27—05:00 Nigel Schuster: Fixed import testcases
574  848066c 2016—11—19T22:24:55—05:00 Nigel Schuster: Moved testcase asset to asset folder
575  53c9206 2016—11—19T22:21:48—05:00 Nigel Schuster: Corrected use of global variable in
      test_globals
576  5fe74a8 2016—11—19T22:21:00—05:00 Nigel Schuster: Fixed expected output for
      test_access_column_cells
577  214ab9d 2016—11—19T22:10:33—05:00 Nigel Schuster: Merge
578  fb31505 2016—11—19T22:08:42—05:00 Nigel Schuster: Passing testcases are in separate
      directory. Output of stats
579  5e39ba7 2016—11—19T21:55:03—05:00 Nigel Schuster: Merge
580  25263fe 2016—11—19T21:51:31—05:00 Nigel Schuster: Removed travis from build, removed
      super verbose output
581  0554ad9 2016—11—19T21:42:28—05:00 Nigel Schuster: Using precise lli version
582  04e5c4a 2016—11—19T18:30:32—05:00 oracleofnj: Add more operators to interpreter
583  e4a190c 2016—11—19T17:14:04—05:00 oracleofnj: Add argument to main and remove
      _expected from filenames
584  7cd2b3a 2016—11—19T16:53:12—05:00 oracleofnj: Merge branch 'master' into fix—test—
      cases
585  d1fddfd 2016—11—19T16:52:48—05:00 oracleofnj: Merge branch 'fix—test—cases' of https
      ://github.com/Neitsch/plt into fix—test—cases
586  36f72a1 2016—11—19T16:49:34—05:00 GitHub: Merge pull request #67 from Neitsch/
      test_cases
587  c46c87b 2016—11—19T16:47:26—05:00 GitHub: Merge branch 'master' into test_cases
588  642ce76 2016—11—19T16:39:50—05:00 Kevin: Fixed helloworld bug
589  ac3d7fa 2016—11—19T16:10:53—05:00 Kevin: Added corresponding AST result for gcd
      function
590  7b6b79e 2016—11—19T14:31:39—05:00 GitHub: Merge branch 'master' into fix—test—cases
591  a9320f3 2016—11—19T14:29:51—05:00 oracleofnj: Merge branch 'master' into fix—test—
      cases
592  24a3625 2016—11—19T14:27:48—05:00 oracleofnj: Add switch tests
593  de262b4 2016—11—19T14:24:39—05:00 GitHub: Merge pull request #60 from Neitsch/box—args
594  75e3f71 2016—11—18T20:39:23—05:00 oracleofnj: Fix parsing errors in test cases
595  4e38757 2016—11—18T16:00:10—05:00 GitHub: Merge branch 'master' into box—args
596  7146dce 2016—11—18T15:59:54—05:00 GitHub: Merge pull request #64 from Neitsch/reorg—
      test
597  f483ac7 2016—11—18T14:10:32—05:00 Kevin: Updated print statement for each test
598  09cb42f 2016—11—18T14:07:39—05:00 oracleofnj: Fix parse difference
599  39634bb 2016—11—18T14:01:21—05:00 oracleofnj: Remove unnecessary files
600  d772725 2016—11—18T14:01:02—05:00 oracleofnj: Make inputs work with interpreter
601  f4456f8 2016—11—18T13:17:25—05:00 GitHub: Merge branch 'master' into test_cases
602  00aafb7 2016—11—18T13:16:08—05:00 Kevin: Renamed inputs folder
603  99db652 2016—11—18T12:51:40—05:00 Kevin: Renamed expected output extension and created
       input folder for test cases
604  2825ada 2016—11—18T12:51:33—05:00 Nigel Schuster: Added branch to build
605  aafabb2 2016—11—18T12:50:56—05:00 Nigel Schuster: Verbose output for travis debug
606  124d61e 2016—11—18T12:44:50—05:00 GitHub: Merge pull request #61 from Neitsch/reorg—
      test
607  82cf599 2016—11—18T12:34:57—05:00 oracleofnj: Modify test script to compare
      interpreter and compiler with expected
608  faecfa1 2016—11—18T01:48:44—05:00 oracleofnj: Fix merge conflict in box_args
```

```
609  41a81ce 2016—11—18T01:40:11—05:00 oracleofnj: Move argument boxing into a function
610  6f63e89 2016—11—18T00:48:07—05:00 GitHub: Merge pull request #59 from Neitsch/hello—
        hello
611  088dc45 2016—11—18T00:29:45—05:00 Nigel Schuster: Merge
612  012caaa 2016—11—18T00:12:40—05:00 GitHub: Merge pull request #58 from Neitsch/copy—
        argv
613  f84757b 2016—11—18T00:02:34—05:00 Nigel Schuster: Removed unneccessary files
614  18fbff1 2016—11—18T00:01:49—05:00 Nigel Schuster: Removed dummy arg reading, added
        printing to interpreter — helloworld TC passes
615  b866da3 2016—11—17T23:31:42—05:00 Nigel Schuster: Made hello world work
616  9463afa 2016—11—17T23:12:41—05:00 oracleofnj: Merge branch 'copy—argv' of https://
        github.com/Neitsch/plt into copy—argv
617  54858ab 2016—11—17T23:11:29—05:00 oracleofnj: Add => infix operator to cut down on all
        the build_struct_gep calls
618  bb11d6d 2016—11—17T23:10:24—05:00 GitHub: Merge branch 'master' into copy—argv
619  e123652 2016—11—17T22:28:12—05:00 oracleofnj: Add byte for zero
620  26a03b7 2016—11—17T22:24:17—05:00 oracleofnj: Add new_string function
621  b8028f9 2016—11—17T20:27:37—05:00 Kevin: Removed files from test folder
622  c85d9b7 2016—11—17T20:25:21—05:00 Kevin: Move testcases to testcases directory
623  f17c6b6 2016—11—17T20:21:38—05:00 Kevin Ye: Complete testcases for List/Range/Function
        /Expression with expected outputs
624  5e63cee 2016—11—17T17:40:31—05:00 GitHub: Merge pull request #54 from Neitsch/
        operation_tests
625  4a4a806 2016—11—17T17:19:13—05:00 GitHub: Merge branch 'master' into operation_tests
626  cafe20e 2016—11—17T17:19:11—05:00 GitHub: Merge pull request #52 from Neitsch/one—main
        —arg
627  4b28df2 2016—11—17T17:17:44—05:00 GitHub: Merge branch 'master' into operation_tests
628  b728e2e 2016—11—17T17:16:20—05:00 GitHub: Merge branch 'master' into one—main—arg
629  d43a87b 2016—11—17T17:15:28—05:00 GitHub: Merge pull request #55 from Neitsch/shell—
        fix
630  b1238a0 2016—11—17T17:08:56—05:00 Nigel Schuster: Shell is not my strength
631  a6cc0ea 2016—11—17T17:05:09—05:00 Nigel Schuster: Screw you bourne shell
632  51fbe67 2016—11—17T16:59:50—05:00 Nigel Schuster: Using bourne shell style redirection
        :
633  3255e1b 2016—11—17T16:38:53—05:00 Ishaan: Modify test suite specs
634  f0ab4d8 2016—11—17T16:38:53—05:00 Ishaan: Moved expected output text files to
        directory
635  06d330c 2016—11—17T16:38:53—05:00 Ishaan: 75% through operator cases
636  e490548 2016—11—17T15:50:35—05:00 GitHub: Merge branch 'master' into one—main—arg
637  a4cf367 2016—11—17T15:50:29—05:00 GitHub: Merge pull request #51 from Neitsch/test—
        script
638  79ee3de 2016—11—17T15:18:58—05:00 oracleofnj: Call main() with first argument <empty>
        in interpreter
639  c4f7437 2016—11—17T14:39:38—05:00 Nigel Schuster: Removed version specific lli
640  7b2236b 2016—11—17T14:35:55—05:00 Nigel Schuster: Fixed if no flag is given
641  e10f656 2016—11—17T14:24:20—05:00 Nigel Schuster: Outputting diff only if —p flag is
        given
642  2d29597 2016—11—17T14:19:30—05:00 Nigel Schuster: Added it as build target
643  7af929a 2016—11—17T14:12:19—05:00 GitHub: Merge pull request #50 from Neitsch/test—
        script
644  6ea43f6 2016—11—17T13:54:55—05:00 Nigel Schuster: Added more env variables to avoid
        copy paste
645  05f27a2 2016—11—17T12:45:11—05:00 Nigel Schuster: Made simple testscript
646  aca43c1 2016—11—17T11:08:11—05:00 Nigel Schuster: Removed accidentally added files
647  9228eac 2016—11—17T04:52:31—05:00 Kevin Ye: Test cases for List of Tests and Range/
        Function/Expression Tests
```

```
648  7feb392 2016—11—17T00:28:53—05:00 GitHub: Merge pull request #48 from Neitsch/
         testing_list
649  6e42afa 2016—11—17T00:27:13—05:00 GitHub: Merge branch 'master' into testing_list
650  e40734b 2016—11—16T23:25:01—05:00 Ishaan: Added more test scenarios
651  41ef578 2016—11—16T17:50:03—05:00 GitHub: Merge pull request #49 from Neitsch/consume—
         command—line—args
652  3cbf089 2016—11—16T17:45:58—05:00 oracleofnj: Fix merge conflict
653  1570836 2016—11—16T16:51:05—05:00 GitHub: Merge pull request #45 from Neitsch/doc
654  a8fbced 2016—11—16T16:38:49—05:00 Nigel Schuster: Fixed minor syntax error
655  c2f37c8 2016—11—16T16:30:43—05:00 Nigel Schuster: Merge
656  2fa73be 2016—11—16T16:05:37—05:00 oracleofnj: Set return code to length of argv[1]
657  bc21af6 2016—11—16T15:54:12—05:00 Ishaan: Added initial testing list
658  cd0d156 2016—11—16T15:50:39—05:00 oracleofnj: Start processing command line args
659  4a1fcac 2016—11—16T13:55:46—05:00 GitHub: Merge pull request #46 from Neitsch/number—
         type
660  f1b481e 2016—11—16T11:04:44—05:00 Nigel Schuster: Added number type that defaults to
         int
661  8944b9a 2016—11—16T00:19:33—05:00 GitHub: Merge pull request #44 from Neitsch/fix—arg
662  92fb7a3 2016—11—15T23:57:37—05:00 Nigel Schuster: Added a little documentation
663  bcbde36 2016—11—15T23:49:07—05:00 GitHub: Merge branch 'master' into fix—arg
664  fa1741a 2016—11—15T23:03:23—05:00 GitHub: Merge pull request #43 from Neitsch/more—
         llvm—gen—js
665  57b2162 2016—11—15T22:39:38—05:00 Nigel Schuster: Using subranges instead of ranges
         everywhere
666  9407677 2016—11—15T22:31:03—05:00 oracleofnj: Add hash table for common functions and
         add dereference—the—range
667  46e1fd5 2016—11—15T21:38:51—05:00 oracleofnj: Eliminate some copy & paste
668  660c049 2016—11—15T20:54:33—05:00 GitHub: Merge pull request #42 from Neitsch/llvm—gen
669  25b23cd 2016—11—15T17:23:54—05:00 Nigel Schuster: Fixed column retrieval for 1x1
670  3f02203 2016—11—15T17:17:02—05:00 Nigel Schuster: Fixed tests
671  26b8fcf 2016—11—15T17:15:08—05:00 Nigel Schuster: Merge
672  e347a87 2016—11—15T17:12:26—05:00 Nigel Schuster: Using more generic flag for values
673  aed28b3 2016—11—15T17:08:07—05:00 oracleofnj: Add is_subrange_1x1
674  cf5cbf0 2016—11—15T14:51:40—05:00 oracleofnj: Merge branch 'llvm—gen' of https://
         github.com/Neitsch/plt into llvm—gen
675  c71d469 2016—11—15T14:51:19—05:00 oracleofnj: Replace String.equal with =
676  4b34abd 2016—11—15T14:41:37—05:00 GitHub: Merge branch 'master' into llvm—gen
677  a80a6d0 2016—11—15T14:41:07—05:00 oracleofnj: Add compile option to main
678  8ad5a19 2016—11—15T14:33:40—05:00 GitHub: Merge pull request #40 from Neitsch/
         interpreter
679  3f0362a 2016—11—15T14:28:44—05:00 GitHub: Merge branch 'master' into interpreter
680  c0c95a2 2016—11—15T14:16:13—05:00 Nigel Schuster: Merge
681  d5f4024 2016—11—15T13:44:44—05:00 Nigel Schuster: Moved failing TCs
682  42fd9ef 2016—11—15T12:21:57—05:00 oracleofnj: Fix bug in import
683  9c567c9 2016—11—15T11:11:30—05:00 Nigel Schuster: Working on imports, fixed most
         testcases
684  aa61ac9 2016—11—15T09:31:42—05:00 Nigel Schuster: Allocating scope object
685  cf1ebf9 2016—11—13T23:09:30—05:00 oracleofnj: Rewrite main to take options; fix bug
         where import didn't know about first filename
686  5749538 2016—11—13T21:59:28—05:00 Nigel Schuster: Added main function
687  d6daff3 2016—11—13T20:26:14—05:00 GitHub: Merge pull request #41 from Neitsch/
         LRM_String_Update
688  0a5d484 2016—11—13T18:45:29—05:00 oracleofnj: Revert "Generating function header"
689  6afe599 2016—11—13T18:44:58—05:00 Ishaan Kolluri: Added changes relating to strings.
690  137d7e2 2016—11—13T18:39:33—05:00 oracleofnj: Merge branch 'interpreter' of https://
         github.com/Neitsch/plt into interpreter
```

691	118bfc5 2016—11—13T18:38:34—05:00 oracleofnj: Allow single slice on RHS; make hashtag work
692	e376270 2016—11—13T17:55:41—05:00 Nigel Schuster: Added type arguments for functions
693	5cfb519 2016—11—13T17:26:23—05:00 Nigel Schuster: Set more types up
694	bf1d8bb 2016—11—13T15:30:35—05:00 Nigel Schuster: Merge branch 'interpreter' of https ://github.com/Neitsch/plt into interpreter
695	f83a0bc 2016—11—13T15:30:28—05:00 Nigel Schuster: Generating function header
696	3addcc8 2016—11—13T14:38:11—05:00 oracleofnj: Make size(expr) an operator instead of built—in function
697	9a74e14 2016—11—13T14:22:44—05:00 oracleofnj: Changing size() to be an operator
698	d6d2eaa 2016—11—13T00:08:41—05:00 oracleofnj: Add closure to interpreter_variable
699	64fba82 2016—11—12T22:38:39—05:00 oracleofnj: Added bsearch to show logic bug
700	66ffdb1 2016—11—12T19:21:07—05:00 oracleofnj: Add alpha version of function calls
701	376b29a 2016—11—12T17:17:23—05:00 oracleofnj: Add string as value type
702	08c61ee 2016—11—12T17:14:47—05:00 oracleofnj: Clean up discrepancies
703	a18d5fc 2016—11—08T11:38:22—05:00 oracleofnj: Fix bug with x[—1]
704	962f812 2016—11—07T23:27:08—05:00 oracleofnj: Refactor scope for interpreter; resolve variables on demand; make selections work properly
705	47bbef1 2016—11—06T22:05:55—05:00 oracleofnj: Minor adjustments to interpreter to work with mapped AST
706	fddc6bc 2016—11—06T18:32:17—05:00 oracleofnj: Eliminate extraneous nulls in JSON
707	ffddb17 2016—11—06T18:15:40—05:00 oracleofnj: Turn statement and function lists into StringMaps
708	6810003 2016—11—05T19:47:57—04:00 oracleofnj: Fix pattern matching warning
709	7107a46 2016—11—05T18:01:34—04:00 oracleofnj: Add function to check range literals for legality at parse time
710	80b13d1 2016—11—05T15:13:10—04:00 oracleofnj: Handle selections better
711	6cbb009 2016—11—04T15:48:58—04:00 oracleofnj: Count to 1,000,000 using tail—recursive versions of List.map and cartesian product
712	9b2252d 2016—11—04T15:25:13—04:00 oracleofnj: Show enter and exit
713	3585e43 2016—11—04T02:21:38—04:00 oracleofnj: See how high it can count recursively
714	38cf541 2016—11—04T02:15:50—04:00 oracleofnj: Get the easy parts of the interpreter working
715	5d81d6e 2016—11—03T17:17:51—04:00 oracleofnj: Start working on interpreter
716	0078cee 2016—11—01T23:40:57—04:00 oracleofnj: Got a non—tail—recursive version of topological sort working
717	85df175 2016—11—01T15:39:10—04:00 oracleofnj: Irrelevant highlighting thing
718	84c719a 2016—11—01T14:39:49—04:00 oracleofnj: Rearrange nested functions
719	557dc4e 2016—11—01T13:50:52—04:00 oracleofnj: Add circular import test case
720	c476798 2016—11—01T13:35:46—04:00 oracleofnj: Fix syntax errors
721	af5a31d 2016—11—01T13:31:49—04:00 GitHub: Merge pull request #37 from Neitsch/import—rec
722	d451cc4 2016—11—01T13:31:33—04:00 GitHub: Merge pull request #38 from Neitsch/import—load
723	02ca24f 2016—11—01T13:30:47—04:00 GitHub: Merge pull request #39 from Neitsch/wild—exc
724	6fa0e39 2016—10—31T16:43:17—04:00 Neitsch: Raising exceptions on certain values
725	e673dca 2016—10—31T15:56:43—04:00 Neitsch: Loading data from all imports
726	6a28c05 2016—10—31T15:40:41—04:00 Neitsch: Recursively looking up dependencies
727	3f28289 2016—10—31T11:53:10—04:00 GitHub: Merge pull request #36 from Neitsch/import—arrange
728	4eaef3b 2016—10—31T11:01:00—04:00 Neitsch: Removed obsolete parts
729	7d7b1e5 2016—10—31T10:59:12—04:00 Neitsch: Added unsorted function, globals and imports
730	7d70af2 2016—10—30T15:23:04—04:00 oracleofnj: Add some explanatory comments
731	40d6b16 2016—10—30T15:03:32—04:00 oracleofnj: More expansion samples
732	af9b01c 2016—10—30T14:48:44—04:00 oracleofnj: Refactor expansion code

```
733  903bc3f 2016-10-30T00:19:10-04:00 oracleofnj: Add test output
734  68b7b03 2016-10-30T00:17:02-04:00 oracleofnj: Add test case
735  a8bdf33 2016-10-30T00:04:05-04:00 oracleofnj: Add LHS slice expansion
736  4ee6fdf 2016-10-29T17:36:17-04:00 oracleofnj: Add output
737  2b8bced 2016-10-29T17:27:22-04:00 oracleofnj: Expand dimension expressions
738  443a818 2016-10-26T16:31:51-04:00 GitHub: Merge pull request #35 from ishaankolluri/
         master
739  9ba3c65 2016-10-26T16:31:00-04:00 Ishaan Kolluri: Add UNIs
740  022e8cd 2016-10-26T16:25:57-04:00 GitHub: Merge pull request #34 from ishaankolluri/
         master
741  808aae5 2016-10-26T16:22:10-04:00 Ishaan Kolluri: Added change to precedence operators
742  0bd9c4a 2016-10-26T15:59:53-04:00 GitHub: Merge pull request #33 from Neitsch/final-
         slicing-comments
743  fb2b382 2016-10-26T15:54:11-04:00 oracleofnj: Thats all for now folks
744  e7020ec 2016-10-26T15:00:11-04:00 GitHub: Merge pull request #32 from Neitsch/final-
         lrm-edits
745  4683f14 2016-10-26T14:48:41-04:00 oracleofnj: Flesh out switch expressions, add
         precedence
746  4b7984a 2016-10-26T11:15:03-04:00 GitHub: Merge pull request #31 from Neitsch/more-lrm
         -edits
747  3d587c5 2016-10-26T11:10:15-04:00 oracleofnj: Incorporate requested edits and a few
         more clarifications
748  0c42b9c 2016-10-26T09:22:08-04:00 GitHub: Merge pull request #30 from ishaankolluri/
         LRM_update
749  cd81040 2016-10-26T03:30:20-04:00 ishaankolluri: Added changes to first half of LRM
750  63fb02b 2016-10-26T02:13:17-04:00 GitHub: Merge pull request #29 from Neitsch/lrm-
         edits
751  0941e96 2016-10-26T02:04:47-04:00 oracleofnj: Rebuild PDF
752  cb04069 2016-10-26T02:04:01-04:00 oracleofnj: Add built in functions
753  4abf638 2016-10-26T01:56:38-04:00 oracleofnj: Add built in functions
754  7661925 2016-10-26T00:04:22-04:00 oracleofnj: Initial comments
755  5932551 2016-10-25T21:30:40-04:00 GitHub: Merge pull request #28 from Neitsch/func-doc
         -fix
756  cc66297 2016-10-25T20:14:27-04:00 Nigel Schuster: Fixed mistakes in functions part of
         the doc
757  b978f00 2016-10-25T13:04:05-04:00 GitHub: Merge pull request #27 from ishaankolluri/
         master
758  125a5bb 2016-10-25T12:49:38-04:00 Ishaan Kolluri: Removed AUX file
759  2e1ea60 2016-10-25T11:30:35-04:00 GitHub: Merge pull request #26 from Neitsch/better-
         regexp
760  84b03ee 2016-10-25T01:22:31-04:00 oracleofnj: Fix let order
761  91b40c5 2016-10-25T01:14:43-04:00 oracleofnj: Improve regexp
762  eb24036 2016-10-24T23:55:38-04:00 GitHub: Merge pull request #23 from Neitsch/file-io
763  991c918 2016-10-24T23:20:12-04:00 oracleofnj: Replace fopen, fclose etc. with open,
         close etc.
764  338faa0 2016-10-24T23:14:30-04:00 oracleofnj: Fix file inclusion and rebuild PDF
765  b24edd3 2016-10-24T23:11:50-04:00 oracleofnj: Merge in expressions section
766  44a1cc5 2016-10-24T23:06:07-04:00 oracleofnj: Merge scanner changes and add regexp to
         properly escape strings
767  2f09a64 2016-10-24T15:52:10-04:00 Kevin: Added the Expression Section 4 to LRM
768  1ea3c28 2016-10-24T15:26:16-04:00 oracleofnj: Merge branch 'master' into file-io
769  ec7cc9c 2016-10-24T15:21:23-04:00 Jared Samet: Replace repetitive code with more
         idiomatic OCaml
770  8cd39ac 2016-10-24T11:05:33-04:00 Kevin: Added string literals to scanner
771  e5d2478 2016-10-24T11:00:39-04:00 Kevin: Added string literals to scanner
772  a692466 2016-10-24T01:09:21-04:00 oracleofnj: Fix tests until strings ready
```

180

```
773  8553a50 2016—10—24T01:08:29—04:00 oracleofnj: Fix tests until string ready
774  0ed4ad7 2016—10—24T00:55:08—04:00 oracleofnj: Add File IO, Entry point and Example to
        LRM
775  71e0b1c 2016—10—23T22:58:21—04:00 oracleofnj: Fix section reference
776  92ac506 2016—10—23T22:39:06—04:00 Ishaan Kolluri: Make small change to data type
        section
777  6abb290 2016—10—23T22:34:42—04:00 oracleofnj: Initial commit for File I/O section
778  67b4b65 2016—10—23T19:30:03—04:00 Nigel Schuster: Reduce eye pain
779  2824ee9 2016—10—23T19:03:24—04:00 GitHub: Merge pull request #20 from Neitsch/samples
780  f8ae543 2016—10—23T18:23:11—04:00 GitHub: Merge branch 'master' into samples
781  13d0896 2016—10—23T18:20:03—04:00 GitHub: Merge pull request #19 from Neitsch/sequence
        —operator
782  e0c702d 2016—10—23T18:17:58—04:00 Neitsch: Fixed .gitignore
783  3a2cd60 2016—10—23T18:16:35—04:00 GitHub: Merge branch 'master' into sequence—operator
784  e42fe94 2016—10—23T18:05:48—04:00 Neitsch: Added code in LRM to test code samples
785  9d2cd17 2016—10—23T17:24:15—04:00 Neitsch: Merge branch 'master' into samples
786  167ddd2 2016—10—23T17:18:35—04:00 Neitsch: Removed test output
787  57319c4 2016—10—23T17:11:13—04:00 oracleofnj: Remove intermediate files
788  53824ea 2016—10—23T17:10:39—04:00 oracleofnj: Flip precedence of —> and ?: (?: is now
        lowest)
789  7dedf93 2016—10—23T17:05:23—04:00 oracleofnj: Add sequence operator to scanner/parser/
        AST
790  9805753 2016—10—23T17:01:31—04:00 GitHub: Merge pull request #17 from Neitsch/make—
        correction
791  e0c7aed 2016—10—23T16:59:33—04:00 Neitsch: Fixed test
792  ec3d682 2016—10—23T16:41:00—04:00 GitHub: Merge branch 'master' into make—correction
793  ea05658 2016—10—23T16:40:24—04:00 Neitsch: Moved sequence file
794  0ca56a0 2016—10—23T16:10:14—04:00 Neitsch: Merge
795  9d1094e 2016—10—23T16:08:59—04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
        config
796  0a28413 2016—10—23T16:08:59—04:00 Neitsch: Completed initial functions section doc
797  0797f32 2016—10—23T16:08:12—04:00 Neitsch: Changed subsection header
798  9df31f7 2016—10—23T16:08:12—04:00 Neitsch: Added dimension section
799  8939903 2016—10—23T16:07:26—04:00 Neitsch: Started working on Functions
800  cae3b37 2016—10—23T16:06:27—04:00 Neitsch: Added dimension section
801  049c95d 2016—10—23T16:06:08—04:00 Neitsch: Started working on Functions
802  84d20b5 2016—10—23T16:01:00—04:00 Neitsch: Comparing sample code with correctly parsed
         code in samples_comp
803  3f015ee 2016—10—23T15:52:01—04:00 GitHub: Merge pull request #18 from Neitsch/grammar—
        bug—fixes
804  7e558c1 2016—10—23T15:44:20—04:00 GitHub: Merge branch 'master' into make—correction
805  edf3dea 2016—10—23T15:44:20—04:00 GitHub: Merge branch 'master' into grammar—bug—fixes
806  d4961eb 2016—10—23T15:43:16—04:00 GitHub: Merge pull request #15 from Neitsch/
        functions—doc
807  0e0bda5 2016—10—23T15:05:42—04:00 GitHub: Merge branch 'master' into functions—doc
808  4652c67 2016—10—23T15:00:35—04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
        config
809  b45718d 2016—10—23T02:27:36—04:00 oracleofnj: Modify grammar to allow [m,n] foo, bar,
        baz;
810  143fcba 2016—10—22T23:23:10—04:00 GitHub: Merge pull request #16 from Neitsch/more—AST
811  a726236 2016—10—22T20:51:27—04:00 oracleofnj: Add comments and sample program
812  8db4098 2016—10—22T19:44:48—04:00 oracleofnj: Fix minor grammar bug
813  80754c3 2016—10—22T18:19:27—04:00 oracleofnj: Hook up scanner and parser
814  660de8c 2016—10—22T13:54:32—04:00 GitHub: Add stuff to the grammar, minor corrections
        (#14)
815  cfe827d 2016—10—21T20:50:51—04:00 Nigel Schuster: Completed initial functions section
```

```
             doc
816  3609366 2016—10—20T21:14:00—04:00 GitHub: Update scanner.mll
817  0d57652 2016—10—20T21:10:27—04:00 Kevin: Fixed bug in scanner
818  1848813 2016—10—20T20:21:49—04:00 Kevin: Made scanner
819  1b610ac 2016—10—20T13:50:22—04:00 Nigel Schuster: Merge
820  acb9b93 2016—10—20T13:44:06—04:00 Nigel Schuster: Changed subsection header
821  b95d039 2016—10—20T13:43:51—04:00 Nigel Schuster: Added dimension section
822  71b93bb 2016—10—20T13:43:09—04:00 Nigel Schuster: Started working on Functions
823  a15772c 2016—10—20T13:38:08—04:00 GitHub: Merge pull request #10 from ishaankolluri/
             LRM
824  dee63c7 2016—10—20T13:26:28—04:00 GitHub: Merge pull request #1 from Neitsch/grammar—
             doc
825  dc93dbf 2016—10—20T13:18:29—04:00 Nigel Schuster: Grammar import
826  4d763cb 2016—10—20T12:44:52—04:00 Ishaan Kolluri: Made refactor and edits to intro
             section of LRM
827  e7443cc 2016—10—20T11:46:54—04:00 Ishaan Kolluri: Merging
828  7542b5d 2016—10—20T11:16:35—04:00 Nigel Schuster: Added dimension section
829  995cf83 2016—10—19T12:28:09—04:00 Nigel Schuster: Started working on Functions
830  40c2a5a 2016—10—19T03:43:06—04:00 ishaankolluri: Initial LRM Commit part 1
831  02a5c17 2016—10—18T18:38:21—04:00 Ishaan Kolluri: Added LRM initial info
832  d8794e9 2016—10—17T19:47:42—04:00 GitHub: Merge pull request #9 from Neitsch/
             documentation
833  70aa1b9 2016—10—16T13:36:23—04:00 Nigel Schuster: Added PDF Latex template
834  5111202 2016—10—14T19:59:45—04:00 GitHub: Added a bunch of stuff to the grammar: (#8)
835  da967e4 2016—10—12T13:24:50—04:00 Jared Samet: CFG Grammar (#6)
836  fea4e4b 2016—10—08T11:42:39—04:00 GitHub: There is no need to constantly build all
             branches. (#2)
837  7a5ccfc 2016—10—08T11:31:31—04:00 Nigel Schuster: Added greeting and newlines (#4)
838  10b17f7 2016—10—08T11:31:08—04:00 GitHub: Imported microc (#5)
839  726456f 2016—09—20T09:45:07—04:00 Nigel Schuster: [test] Add sample greeting to repo
             (#3)
840  9a2183d 2016—09—15T18:44:00—04:00 Nigel Schuster: Added merlin config
841  163e176 2016—09—14T18:51:53—04:00 Nigel Schuster: Moved whole build to script
842  d401eea 2016—09—14T18:43:58—04:00 Nigel Schuster: Added oasis opam package
843  ba7fd9c 2016—09—14T18:38:58—04:00 Nigel Schuster: Added ocaml configure (maybe this
             helps travis)
844  a461eae 2016—09—14T18:26:10—04:00 Nigel Schuster: Configuring opam environment for
             travis
845  ba2df2f 2016—09—14T18:19:26—04:00 Nigel Schuster: Added ocaml native compiler to apt
             package list
846  a8e5958 2016—09—14T17:24:36—04:00 Nigel Schuster: Added some more (possibly necessary
             opam packages
847  c54f5e3 2016—09—14T17:18:32—04:00 Nigel Schuster: Missed opam option
848  b10adf0 2016—09—14T17:13:57—04:00 Nigel Schuster: Fixed opam install
849  124f7f3 2016—09—14T17:08:09—04:00 Nigel Schuster: Fixed YML error
850  4909fa8 2016—09—14T17:03:54—04:00 Nigel Schuster: Using avsm source
851  4b24046 2016—09—14T16:58:33—04:00 Nigel Schuster: Allow sudo
852  e7b50db 2016—09—14T16:56:57—04:00 Nigel Schuster: Fixed setup order
853  f6d7ac4 2016—09—14T16:50:02—04:00 Nigel Schuster: Manually installing apt packages
854  f4084ab 2016—09—14T16:40:55—04:00 Nigel Schuster: Test commit
855  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.