Extend Language Final Report

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

December 17, 2016

Contents

1. Introduction

Extend is a declarative programming language meant to support spreadsheet-like functionality. It contains features such as complex calculations on large inputs of data, side-effect free and immutable values, 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 decisions.

- In order to support more complex data structures, we must allow complex content within cells as opposed 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 fairly forgiving to the user.

Microsoft Excel and Google Sheets find issues at scale when users need to process more and more complicated calculations on several different sets of data, sometimes at scheduled intervals. Extend was conceptualized as a standalone application that removes the manual element of entering new inputs. It brings the best of spreadsheets and computation into one product.

Complex Calculations Across Many Inputs

Extend is spiritually closer in behavior to Microsoft Excel than other conventional programming languages. Extend can nest basic operations in cells of a range, Extend's proprietary data type, and apply mathematical operations on both individual cells of a range and the entire range itself. A dependency graph is evaluated at runtime to optimize and execute these calculations.

Flexibility

Extend, as a declarative language, allows you to keep dimensional values potentially variable until runtime, and handles some errors with the "empty" keyword, which persists throughout calculation instead of

crashing the program. It supports a range of standard library functions that are additionally supported in conventional spreadsheet technology as well.

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 example_source_file.xtnd
```

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

2.3 Illustrating the Benefits of Extend

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:

The Excel user must manually input the data, and additionally make space for the intermediate steps of the calculation. As the data becomes more diverse and the problem becomes more complicated, more work is required. 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 particularly interesting 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.

Performing a delimiter 'join' like the above can be performed in a simple program in Extend without knowing the size of the row.

```
1   main(args){
2     bar := {"Hello", "Goodbye", "Hello Again"};
3     str := ", ";
4     return foo(bar, str);\\ prints "Hello, Goodbye, Hello Again"
5   }
6
7   foo([1,n] colrange, str){
8     [1,n] baz;
9     baz[0,0] = #colrange;
10    baz[0,1:] = baz[0,[-1]] + str + colrange[0,[0]];
11    return print_endline(baz[0,-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.

2.4   Writing Extend Code - The Basics

Extend code can be written in a file that contains the conventional `.xtnd` extension. It consists of optional import statements and global variables, and an optional set of functions. Runnable Extend programs must contain a `main` function, and other functions can be written into the program as well.

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

Functions

Functions are commonplace in Extend. They are declared with the syntax `function_name([optional_dimensions] function_arguments){...}`. Below is the syntax of the `main` function, which is needed to run Extend code, within a simple Extend program.

```
1    main(args){
2        return 0;
3    }
```

The return type of a function is a **range** of flexible size. **Ranges** will be discussed in a later section of this chapter. A function is composed of a series of statements, concluding with the return statement. Note that the **return** statement is always the last statement in the function.

Function Parameters

Function parameters consist of zero or more ranges, signed with an optional dimension. If the arguments have been written with dimensions, those dimensions will be verified at runtime.

```
1        foo([m,n] arg){
2          return m * n; \\ m and n initialized through arg1
3        }
4
5        bar([1,1] arg){
6          return 0; \\ 1 by 1 ranges should be primitive data types. If arg is not, a
                 runtime error will be thrown
7        }
```

Adjusting to Extend's Declarative Behavior

An important aspect of Extend is that the concept of a simple statement, barring the `return` statement, does not exist. In an Extend function, there are only declarations, formula assignments, and a return expression. As a declarative language, Extend examines the `return` statement, and simply evaluates the lines that will help get that return value. All other lines are ignored.

The below Extend source file will not compile.

```
1    main(args){
2      foo := "Hello World!";
3      print_endline(foo);\\ Cannot have this line. Must equal some formula.
4      return foo;
5    }
```

The below file compiles and prints successfully.

```
1      main(args){
2        foo := "Hello World!";
3        return print_endline(foo); \\ print_endline is required in the return statement,
             and therefore runs
4      }
```

As illustrated, Extend only evaluates what is needed to produce the value required by `return`. Any non-essential declarations or formula assignments will be ignored by the program. If the user attempts to write statements like `print_endline("Hello")` by itself, the program will not compile.

Primitive Data Types

Extend has three primitive data types: Numbers, Strings, and `empty`. An example of each is shown below.

```
1      bar := 5; \\ Assigns bar to a number
2      foo := "Hello World"; \\ Assigns foo to a string
3      bar = empty; \\ Reassigns bar to empty
```

Note that we declare a variable and assign it a value together with `:=`. If the variable already exists, `=` will suffice. More information about Extend's primitive data types can be found in the Language Reference Manual.

The typeof function

Extend offers a `typeof(expr)` function, which takes and expression and returns Number, String, Range, or Empty.

Operators

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

Arithmetic Operators

- Unary Operations: `-`

- Binary Operations: `**, *, /, %, +, -`

Bitwise Operators

- Unary Operations: `~`

- Binary Operations: `«, », &, |, ^`

Boolean Operators

- Unary Operations: `!`

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

String Concatenation

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

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

Conditionals

There are two types of conditionals: the ternary conditional and a `switch` expression.

The Ternary Conditional

The two ways to write the ternary condition are as follows:

```
condition ?  expr_if_true :  expr_if_false
```

or:

```
if(conditional, expr_if_true, expr_if_false)
```

The Switch Expression

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

```
1    switch_func(foo){
2      return switch(foo){
3        case 1: "Hello, one!";
4        case 2: "Hello, two!";
5        default: "Hello?";
6      };
7    }
```

Ranges

In Extend, `variables` are composed of cells. Under the hood, each cell points to a structure that may contain either primitive values or another `range`. This range can point to anonymous variables that are dereferenced in the aforementioned fashion as many times as needed to reveal the primitive value of the cell.

Ranges are an incredibly important aspect of Extend. A range is composed of rows and columns of values. Each value can contain a formula that can evaluate to another range or one of the aforementioned three data types - number, string, or `empty`. Note that **1 by 1** ranges are automatically dereferenced to their contents in Extend.

```
1    [1,2] foo; \\ Declares range of dimensions 1 by 2
2    [1,3] bar := 4; \\ Declares range of dimensions 1 by 3, and assigns each cell to 4
```

Ranges can also be declared as range literals. Rows are delineated by ; and columns by ,.

```
1    foo := {"Hello", 1, 2; 3, 4, 5; 6, 7, 8} \\ Three rows, three columns.
```

Range Slicing & Selection

Python-style array-slicing syntax can be applied to ranges, which will return a subrange based on either absolute or relative indexing. All indices are zero-based.

```
1    foo[0,2] \\ The cell in the first row, third column
2    foo[0,:] \\ The range of cells in the first row
3    foo[0,[1]] \\ The range in the column that is 1 column right of the left—hand—
        side cell.
4    foo[,] \\ Cell in first row, first column if 1 by 1. If not, then relative first
        row and relative first column
```

More examples and detail can be found in the next chapter.

The Hash Operator

A common case for ranges in Excel is to perform calculations on specific cells. For example, `foo[,]` is commonly used to retrieve the cell that is being calculated on. Since this is a popular use case, the `#` operator will perform the same functionality.

Application on Ranges

Extend, in the vein of spreadsheets, allows the programmer to apply functions cell-wise on a range. Using the `#` operator, we can perform cell-wise multiplication across two ranges.

```
1       foo([m,n] arg1, [m.n] arg2){
2         [m,n] bar := #arg1 * #arg2; \\ Multiplies the cell in arg1 with the
              corresponding cell in arg2.
3         return bar;
4       }
```

This is an incredibly powerful aspect of Extend. Make sure to study it well!

Range Attribute Functions

Extend has the `row()` and `column()` functions, which respectively return the row and column of the cell that is being calculated at that point in time. There is also a `size(expr)` function, which returns a 1 by 2 range; the first cell contains the number of rows, and the second cell contains the number of columns.

Precedence Expressions

Precedence Expressions are used to force the evaluation of one expression before another. This is used when the order of operation is required for functions with side effects, such as the ones housed in Extend's standard library. The `->` operator is used to denote precedence.

```
1     return print_func(4) -> 0; \\ print_func prints 4 to stdout before returning 0
```

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/helloworld.xtnd"
```

2.5 Standard Library Functions

Extend offers an assortment of standard library functions. Extend imports `stdlib.xtnd`, which has aggregated all the standard library functions for the user's disposal.

While their usage will be covered in more length in the Language Reference Manual, here are some of the more useful standard library functions to remember.

Basic Functions

The toString() Function

The `toString()` function takes a 1 by 1 range and renders its value as a string. This will return one of the primitive data types.

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

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 write(STDOUT, toString(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     }
```

Plotting

Extend also offers the ability to export plots to a PNG file with the `plot` function.

```
1    TODO: Demonstrate usage here.
```

3. Language Reference Manual

3.1 Introduction to Extend

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

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

3.2 Structure of an Extend Program

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

Import Statements

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

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

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

Function Definitions

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

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

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

Global Variables

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

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

External Library Declarations

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

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

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

`main` function

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

Scoping and Namespace

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

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

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

3.3   Types and Literals

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

Primitive Data Types

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

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

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

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

Ranges

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

Range Literals

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

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

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

3.4   Expressions

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

Arithmetic Operators

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

Operator	Description	Definition
`~`	`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. The arguments are evaluated from left to right before the function is called. For more detail, see section 3.5.

Range Expressions

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

Slices

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

Selections

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

Corresponding Cell

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

Selection Examples

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

Precedence Expressions

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

3.5   Functions

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

Format

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

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

Variable Declarations

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

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

Formula Assignment

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

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

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

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

Combined Variable Declaration and Formula Assignment

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

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

Formula Assignment Errors

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

Parameter Declarations

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

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

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

Application on Ranges

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

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

This function normalizes a column vector to have unit norm:

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

Lazy Evaluation and Circular References

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

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

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

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

External Libraries

Using the following library declaration:

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

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

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

3.6   Introduction to Extend

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

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

3.7   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 }
```

3.8   Standard Library Reference

3.9   Example Program

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

4. Project Plan

4.1 Meetings

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

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

4.2 Development Workflow
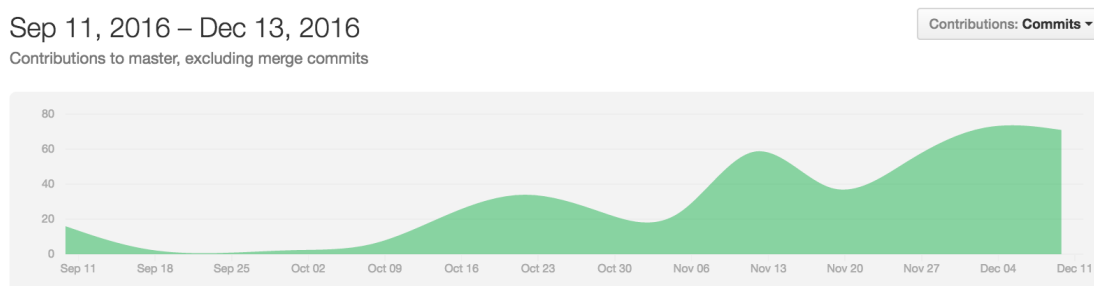
Github & Travis CI

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

An important aspect of development for us was continuous integration. We used Travis CI to trigger project builds on each pull request, 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   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
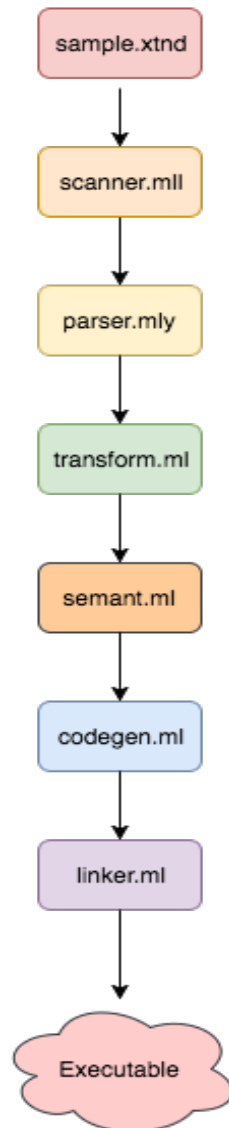
5.1   The Extend Compiler

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 expands compact statements in the Extend syntax tree into statements with equivalent functionality, but reduced breadth. This step is done to preserve the convenience for the user, but revert the code later into a form that is easier for the compiler to chew on.

The Semantic Analyzer

The semantic analyzer 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.

The Code Generator

Provided that the program was deemed legal by `transform.ml`, the code generator will take the program definition in the abstract syntax tree and generate the appopriate LLVM IR to turn it into a functional program. Instructions to allocate memory, interact with external functions, and platform optimization can be found here.

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.

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. Extend Code Listing

7.1 scanner.mll

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

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

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

7.2   parser.mly

```
1   /* Ocamlyacc parser for Extend */
2
3   %{
4   open Ast
5   %}
6
7   %token LSQBRACK RSQBRACK LPAREN RPAREN LBRACE RBRACE HASH
8   %token COLON COMMA QUESTION IF GETS ASN SEMI PRECEDES
9   %token SWITCH CASE DEFAULT SIZE TYPEOF ROW COLUMN
10  %token PLUS MINUS TIMES DIVIDE MOD POWER LSHIFT RSHIFT
11  %token EQ NOTEQ GT LT GTEQ LTEQ
12  %token LOGNOT LOGAND LOGOR
13  %token BITNOT BITXOR BITAND BITOR
14  %token EMPTY RETURN IMPORT GLOBAL EXTERN
15  %token DEBUG
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) }
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    | DEBUG LPAREN expr RPAREN { Debug($3) }
122    | expr PRECEDES expr  { Precedence($1, $3) }
123    | LPAREN expr RPAREN  { $2 }
124    | ID                  { Id($1) }
125    | LIT_INT             { LitInt($1) }
126    | LIT_FLOAT           { LitFlt($1) }
127    | LIT_STRING          { LitString($1) }
128    | EMPTY               { Empty }
129
130  op_expr:
131      expr PLUS expr      { BinOp($1, Plus, $3) }
132    | expr MINUS expr     { BinOp($1, Minus, $3) }
133    | expr TIMES expr     { BinOp($1, Times, $3) }
134    | expr DIVIDE expr    { BinOp($1, Divide, $3) }
135    | expr MOD expr       { BinOp($1, Mod, $3) }
136    | expr POWER expr     { BinOp($1, Pow, $3) }
137    | expr LSHIFT expr    { BinOp($1, LShift, $3) }
138    | expr RSHIFT expr    { BinOp($1, RShift, $3) }
139    | expr LOGAND expr    { BinOp($1, LogAnd, $3) }
140    | expr LOGOR expr     { BinOp($1, LogOr, $3) }
141    | expr BITXOR expr    { BinOp($1, BitXor, $3) }
142    | expr BITAND expr    { BinOp($1, BitAnd, $3) }
143    | expr BITOR expr     { BinOp($1, BitOr, $3) }
144    | expr EQ expr        { BinOp($1, Eq, $3) }
145    | expr NOTEQ expr     { UnOp(LogNot,(BinOp($1, Eq, $3))) }
146    | expr GT expr        { BinOp($1, Gt, $3) }
147    | expr LT expr        { BinOp($1, Lt, $3) }
148    | expr GTEQ expr      { BinOp($1, GtEq, $3) }
149    | expr LTEQ expr      { BinOp($1, LtEq, $3) }
150    | SIZE LPAREN expr RPAREN { UnOp(SizeOf, $3) }
151    | TYPEOF LPAREN expr RPAREN { UnOp(TypeOf, $3) }
152    | ROW LPAREN RPAREN       { UnOp(Row, Empty)}
153    | COLUMN LPAREN RPAREN    { UnOp(Column, Empty)}
```

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

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

7.3 ast.ml

```
1  type op       = Plus | Minus | Times | Divide | Mod | Pow |
2                  LShift | RShift | BitOr | BitAnd | BitXor |
3                  Eq | Gt | GtEq | Lt | LtEq | LogAnd | LogOr
4  type unop     = Neg | LogNot | BitNot | SizeOf | TypeOf | Row | Column | Truthy
```

```
 5
 6  type expr      = LitInt of int |
 7                   LitFlt of float |
 8                   LitString of string |
 9                   LitRange of (expr list) list |
10                   Id of string |
11                   Empty |
12                   BinOp of expr * op * expr |
13                   UnOp of unop * expr |
14                   Ternary of expr * expr * expr |
15                   Switch of expr option * case list * expr |
16                   Call of string * expr list |
17                   Selection of expr * sel |
18                   ReducedTernary of string * string * string |
19                   Precedence of expr * expr |
20                   Debug of 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   | Debug(e) -> string_of_expr e
155
156 and string_of_case (el, e) =
157     "{\"Cases\": " ^ string_of_list (Exprs el) ^ ", " ^
158      "\"expr\": " ^ string_of_expr e ^ "}"
159
160 and string_of_sel (s1, s2) =
161     "{\"slice1\": " ^ string_of_slice s1 ^ ", \"slice2\": " ^ string_of_slice s2 ^ "}"
162
163 and string_of_slice = function
164     None -> "null"
165   | Some (start_idx, end_idx) -> "{\"start\": " ^ string_of_index start_idx ^ ", \"end
        \": " ^ string_of_index end_idx ^ "}"
166
```

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

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

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

7.4   transform.ml

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

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

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

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

```
175          name = f.name;
176          params = f.params;
177          raw_asserts = [new_assert];
178          body = new_sizevars @ size_inits @ retval_inits @ assert_inits @
                   expand_stmt_list f.name f.body;
179          ret_val = (fst f.ret_val, new_retval)
180        } in
181    (imports, expand_stmt_list "global" globals, List.map expand_function functions,
          externs);;
182
183  let create_maps (imports, globals, functions, externs) =
184    let vd_of_vi = function
185      (*  vd_of_vi—— Take a bare Varinit from the previous transformations
186          and return a (string, variable) pair      *)
187        Varinit((Some r, Some c), [(v, None)]) —> (v, {
188          var_rows = (match r with
189                LitInt(1) —> DimOneByOne
190              | Id(s) —> DimId(s)
191              | _ —> raise (LogicError("Unrecognized expression for rows of " ^ v)));
192          var_cols = (match c with
193                LitInt(1) —> DimOneByOne
194              | Id(s) —> DimId(s)
195              | _ —> raise (LogicError("Unrecognized expression for rows of " ^ v)));
196          var_formulas = [];
197        })
198      | _ —> raise (LogicError("Unrecognized format for post—desugaring Varinit")) in
199
200    let add_formula m = function
201          Varinit(_,_) —> m
202        | Assign(var_name, (Some (Some row_start, row_end), Some (Some col_start, col_end
            )), Some e) —>
203          if StringMap.mem var_name m
204          then (let v = StringMap.find var_name m in
205                StringMap.add var_name {v with var_formulas = v.var_formulas @ [{
206                    formula_row_start = row_start;
207                    formula_row_end = row_end;
208                    formula_col_start = col_start;
209                    formula_col_end = col_end;
210                    formula_expr = e;
211                  }]} m)
212          else raise (UnknownVariable(string_of_stmt (Assign(var_name, (Some (Some
                row_start, row_end), Some (Some col_start, col_end)), Some e))))
213        | Assign(a) —> raise (LogicError("Unrecognized format for post—desugaring Assign:
              " ^ string_of_stmt (Assign(a)))) in
214
215    let vds_of_stmts stmts =
216      let is_varinit = function Varinit(_,_) —> true | _ —> false in
217      let varinits = List.filter is_varinit stmts in
218      let vars_just_the_names = map_of_list (List.map vd_of_vi varinits) in
219      List.fold_left add_formula vars_just_the_names stmts in
220
221    let fd_of_raw_func f = (f.name, {
222        func_params = f.params;
223        func_body = vds_of_stmts f.body;
224        func_ret_val = f.ret_val;
225        func_asserts = f.raw_asserts;
```

```
226        }) in
227
228      let tupleize_library (Library(lib_name, lib_fns)) =
229        List.map (fun ext_fn -> (ext_fn.extern_fn_name, {ext_fn with extern_fn_libname =
                lib_name})) lib_fns in
230
231      (vds_of_stmts globals,
232       map_of_list (List.map fd_of_raw_func functions),
233       map_of_list (List.concat (List.map tupleize_library externs)))
234
235    let single_formula e = {
236      formula_row_start = DimensionStart;
237      formula_row_end = Some DimensionEnd;
238      formula_col_start = DimensionStart;
239      formula_col_end = Some DimensionEnd;
240      formula_expr = e;
241    }
242
243    let ternarize_exprs (globals, functions, externs) =
244      let rec ternarize_expr lhs_var = function
245          BinOp(e1, LogAnd, e2) ->
246          let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
247          let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
248          (Ternary(UnOp(Truthy,new_e1), UnOp(Truthy,new_e2), LitInt(0)), new_e1_vars @
                new_e2_vars)
249        | BinOp(e1, LogOr, e2) ->
250          let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
251          let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
252          (Ternary(UnOp(Truthy,new_e1), LitInt(1), UnOp(Truthy,new_e2)), new_e1_vars @
                new_e2_vars)
253        | BinOp(e1, op, e2) ->
254          let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
255          let (new_e2, new_e2_vars) = ternarize_expr lhs_var  e2 in
256          (BinOp(new_e1, op, new_e2), new_e1_vars @ new_e2_vars)
257        | UnOp(op, e) ->
258          let (new_e, new_e_vars) = ternarize_expr lhs_var e in
259          (UnOp(op, new_e), new_e_vars)
260        | Ternary(cond, e1, e2) ->
261          let (new_cond, new_cond_vars) = ternarize_expr lhs_var cond in
262          let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
263          let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
264          (Ternary(new_cond, new_e1, new_e2), new_cond_vars @ new_e1_vars @ new_e2_vars)
265        | Call(fname, args) ->
266          let new_args_and_vars = List.map (ternarize_expr lhs_var) args in
267          (Call(fname, (List.map fst new_args_and_vars)), List.concat (List.map snd
                new_args_and_vars))
268        | Selection(e, (sl1, sl2)) ->
269          let (new_e, new_e_vars) = ternarize_expr lhs_var e in
270          let (new_sl1, new_sl1_vars) = ternarize_slice lhs_var sl1 in
271          let (new_sl2, new_sl2_vars) = ternarize_slice lhs_var sl2 in
272          (Selection(new_e, (new_sl1, new_sl2)), new_e_vars @ new_sl1_vars @ new_sl2_vars)
273        | Precedence(e1, e2) ->
274          let (new_e1, new_e1_vars) = ternarize_expr lhs_var e1 in
275          let (new_e2, new_e2_vars) = ternarize_expr lhs_var e2 in
276          (Precedence(new_e1, new_e2), new_e1_vars @ new_e2_vars)
277        | Switch(cond, cases, dflt) ->
```

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

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

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

7.5   semant.ml

```
1 open Ast
2
3 exception IllegalExpression of string;;
```

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

```
53                              string_of_int num_args ^ " arguments " ^ "but the signature
                                   specifies "
54                              ^ string_of_int signature_args) ;
55           raise(WrongNumberArgs(context ^ "," ^ called_fname)))
56         else () in
57
58     let rec check_expr fname symbols = function
59         BinOp(e1,_,e2) -> check_expr fname symbols e1 ; check_expr fname symbols e2
60       | UnOp(_, e) -> check_expr fname symbols e
61       | Ternary(cond, e1, e2) -> check_expr fname symbols cond ; check_expr fname
             symbols e1 ; check_expr fname symbols e2
62       | ReducedTernary(s1, s2, s3) -> check_expr fname symbols (Id(s1)) ; check_expr
             fname symbols (Id(s2)) ; check_expr fname symbols (Id(s3))
63       | Id(s) -> if StringMap.mem s symbols then () else raise(UnknownVariable(fname ^
             "(): " ^ s))
64       | 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
65       | Switch(None, cases, dflt) -> List.iter (fun c -> check_case fname symbols c)
             cases ; check_expr fname symbols dflt
66       | Call(called_fname, args) ->
67         check_call fname called_fname (List.length args) ;
68         List.iter (fun a -> check_expr fname symbols a) args
69       | Selection(e, (sl1, sl2)) -> check_expr fname symbols e ; check_slice fname
             symbols sl1 ; check_slice fname symbols sl2
70       | Precedence(e1, e2) -> check_expr fname symbols e1 ; check_expr fname symbols e2
71       | Debug(e) -> check_expr fname symbols e;
72       | LitInt(_) | LitFlt(_) | LitRange(_) | LitString(_) | Empty -> ()
73     and check_case fname symbols (conds, e) = List.iter (fun c -> check_expr fname
         symbols c) conds ; check_expr fname symbols e
74     and check_slice fname symbols = function
75         None -> ()
76       | Some (i1, i2) -> check_index fname symbols i1 ; check_index fname symbols i2
77     and check_index fname symbols = function
78         Some Abs(e) -> check_expr fname symbols e
79       | Some Rel(e) -> check_expr fname symbols e
80       | _ -> () in
81     let check_formula fname symbols f =
82       check_index fname symbols (Some f.formula_row_start) ;
83       check_index fname symbols f.formula_row_end ;
84       check_index fname symbols (Some f.formula_col_start) ;
85       check_index fname symbols f.formula_col_end ;
86       check_expr fname symbols f.formula_expr in
87     let check_dim fname symbols = function
88         DimOneByOne -> ()
89       | DimId(s) -> check_expr fname symbols (Id(s)) in
90     let check_variable fname symbols v =
91       check_dim fname symbols v.var_rows ;
92       check_dim fname symbols v.var_cols ;
93       List.iter (fun f -> check_formula fname symbols f) v.var_formulas in
94     let check_variables context symbols vars =
95       StringMap.iter (fun _ v -> check_variable context symbols v) vars in
96
97     let check_function fname f =
98       if StringMap.mem fname externs then raise(DuplicateDefinition(fname ^ "() is
             defined as both an external and local function")) else ();
99       let locals = f.func_body in
```

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

7.6   codeGenTypes.ml

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

```
40  };;
41
42  type scope_field_type = VarDefn | VarInst | VarNum | ScopeRefCount | FunctionParams
43  let scope_field_type_index = function
44      VarDefn -> 0
45    | VarInst -> 1
46    | VarNum -> 2
47    | ScopeRefCount -> 3
48    | FunctionParams -> 4
49
50  type value_field_flags = Empty | Number | String | Range
51  let value_field_flags_index = function
52      Empty -> 0
53    | Number -> 1
54    | String -> 2
55    | Range -> 3
56  let int_to_type_array = [|"Empty"; "Number"; "String"; "Range"|]
57
58  type value_field = Flags | Number | String | Subrange
59  let value_field_index = function
60      Flags -> 0
61    | Number -> 1
62    | String -> 2
63    | Subrange -> 3
64
65  type var_defn_field = Rows | Cols | NumFormulas | Formulas | OneByOne | VarName
66  let var_defn_field_index = function
67      Rows -> 0
68    | Cols -> 1
69    | NumFormulas -> 2
70    | Formulas -> 3
71    | OneByOne -> 4
72    | VarName -> 5
73
74  type formula_field  = FromFirstRow | RowStartNum | ToLastRow | RowEndNum |
        FromFirstCols | ColStartNum | ToLastCol | ColEndNum | IsSingleRow | IsSingleCol |
        FormulaCall
75  let formula_field_index = function
76      FromFirstRow -> 0
77    | RowStartNum -> 1
78    | ToLastRow -> 2
79    | RowEndNum -> 3
80    | FromFirstCols -> 4
81    | ColStartNum -> 5
82    | ToLastCol -> 6
83    | ColEndNum -> 7
84    | IsSingleRow -> 8
85    | IsSingleCol -> 9
86    | FormulaCall -> 10
87
88  type var_instance_field = Rows | Cols | NumFormulas | Formulas | Closure | Values |
        Status
89  let var_instance_field_index = function
90      Rows -> 0
91    | Cols -> 1
92    | NumFormulas -> 2
```

```
 93    | Formulas -> 3
 94    | Closure -> 4
 95    | Values -> 5
 96    | Status -> 6
 97
 98  type var_instance_status_flags = NeverExamined | Calculated | InProgress
 99  let var_instance_status_flags_index = function
100      NeverExamined -> 0
101    | Calculated -> 2
102    | InProgress -> 4
103
104  type subrange_field = BaseRangePtr | BaseOffsetRow | BaseOffsetCol | SubrangeRows |
        SubrangeCols
105  let subrange_field_index = function
106      BaseRangePtr -> 0
107    | BaseOffsetRow -> 1
108    | BaseOffsetCol -> 2
109    | SubrangeRows -> 3
110    | SubrangeCols -> 4
111
112  type dimensions_field = DimensionRows | DimensionCols
113  let dimensions_field_index = function
114      DimensionRows -> 0
115    | DimensionCols -> 1
116
117  type string_field = StringCharPtr | StringLen | StringRefCount
118  let string_field_index = function
119      StringCharPtr -> 0
120    | StringLen -> 1
121    | StringRefCount -> 2
122
123  type rhs_index_field = RhsExprVal | RhsIndexType
124  let rhs_index_field_index = function
125      RhsExprVal -> 0
126    | RhsIndexType -> 1
127
128  type rhs_index_type_flags = RhsIdxAbs | RhsIdxRel | RhsIdxDimStart | RhsIdxDimEnd
129  let rhs_index_type_flags_const = function
130      RhsIdxAbs -> 0
131    | RhsIdxRel -> 1
132    | RhsIdxDimStart -> 2
133    | RhsIdxDimEnd -> 4 (* No 3 *)
134
135  type rhs_slice_field = RhsSliceStartIdx | RhsSliceEndIdx
136  let rhs_slice_field_index = function
137      RhsSliceStartIdx -> 0
138    | RhsSliceEndIdx -> 1
139
140  type rhs_selection_field = RhsSelSlice1 | RhsSelSlice2
141  let rhs_selection_field_index = function
142      RhsSelSlice1 -> 0
143    | RhsSelSlice2 -> 1
144
145  let setup_types ctx =
146    let var_instance_t = Llvm.named_struct_type ctx "var_instance" (*Range struct is a 2
          D Matrix of values*)
```

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

```
200        int_t(*numFormulas*);
201        resolved_formula_p(*formula with resolved dimensions*);
202        extend_scope_p(*scope that contains all variables of a function*);
203        value_p_p(*2D array of cell values*);
204        char_p(*2D array of calculation status for each cell*);
205        char_p(*Name*);
206      ]) false
207    and _ = Llvm.struct_set_body var_defn_t (Array.of_list [
208        int_t(*Rows*);
209        int_t(*Cols*);
210        int_t(*Number of formulas*);
211        formula_p;
212        char_t(*Is one by one range*);
213        char_p(*Name*);
214      ]) false
215    and _ = Llvm.struct_set_body formula_t (Array.of_list [
216        char_t (*from First row*);
217        int_t (*row Start num*);
218        char_t (*to last row*);
219        int_t (*row end num*);
220        char_t (*from first col*);
221        int_t (*col start*);
222        char_t (*to last col*);
223        int_t (*col end num*);
224        char_t (* is single row *);
225        char_t (* is single col *);
226        formula_call_p (*formula to call*);
227      ]) false
228    and _ = Llvm.struct_set_body extend_scope_t (Array.of_list [
229        var_defn_p(*variable definitions*);
230        var_instance_p_p(*variable instances*);
231        int_t(*number of variables*);
232        int_t(*reference count*);
233        Llvm.pointer_type value_p;
234      ]) false
235    and _ = Llvm.struct_set_body subrange_t (Array.of_list [
236        var_instance_p(*The target range*);
237        int_t(*row offset*);
238        int_t(*column offset*);
239        int_t(*row count*);
240        int_t(*column count*)
241      ]) false
242    and _ = Llvm.struct_set_body value_t (Array.of_list [
243        flags_t (*First bit indicates whether it is an int or a range*);
244        number_t (*Numeric value of the cell*);
245        string_p (*String value of the cell if applicable*);
246        subrange_p (*Range value of the cell if applicable*);
247        (*float_t (Double value of the cell*)
248      ]) false
249    and _ = Llvm.struct_set_body string_t (Array.of_list [
250        char_p (*Pointer to null—terminated string*);
251        long_t (*Length of string*);
252        int_t (*Reference count*)
253      ]) false
254    and _ = Llvm.struct_set_body dimensions_t (Array.of_list [int_t; int_t]) false in
255    {
```

```
256      var_instance_t = var_instance_t;
257      value_t = value_t;
258      subrange_t = subrange_t;
259      resolved_formula_t = resolved_formula_t;
260      dimensions_t = dimensions_t;
261      number_t = number_t;
262      string_t = string_t;
263      extend_scope_t = extend_scope_t;
264      formula_t = formula_t;
265      formula_call_t = formula_call_t;
266
267      var_defn_t = var_defn_t;
268      var_defn_p = var_defn_p;
269      var_instance_p = var_instance_p;
270      subrange_p = subrange_p;
271      value_p = value_p;
272      resolved_formula_p = resolved_formula_p;
273      string_p = string_p;
274      char_p = char_p;
275      extend_scope_p = extend_scope_p;
276      formula_p = formula_p;
277      formula_call_p = formula_call_p;
278
279      var_instance_p_p = var_instance_p_p;
280
281      int_t = int_t;
282      long_t = long_t;
283      float_t = float_t;
284      flags_t = flags_t;
285      bool_t = bool_t;
286      char_t = char_t;
287      void_t = void_t;
288      char_p_p = char_p_p;
289      string_p_p = string_p_p;
290
291      rhs_index_t = rhs_index_t;
292      rhs_slice_t = rhs_slice_t;
293      rhs_selection_t = rhs_selection_t;
294      rhs_index_p = rhs_index_p;
295      rhs_slice_p = rhs_slice_p;
296      rhs_selection_p = rhs_selection_p;
297    }
```

7.7   codegen.ml

```
 1  (* Extend code generator *)
 2
 3  open Ast
 4  open Semant
 5  open CodeGenTypes
 6  exception NotImplemented
 7
 8  let runtime_functions = Hashtbl.create 20
 9
10  let (=>) struct_ptr elem = (fun val_name builder ->
```

```
11      let the_pointer = Llvm.build_struct_gep struct_ptr elem "the_pointer" builder in
12      Llvm.build_load the_pointer val_name builder);;
13
14  let ($>) val_to_store (struct_ptr, elem)  = (fun builder ->
15      let the_pointer = Llvm.build_struct_gep struct_ptr elem "" builder in
16      Llvm.build_store val_to_store the_pointer builder);;
17
18  (* from http://stackoverflow.com/questions/243864/what-is-the-ocaml-idiom-equivalent-
        to-pythons-range-function without the infix *)
19  let zero_until i =
20    let rec aux n acc =
21      if n < 0 then acc else aux (n-1) (n :: acc)
22    in aux (i-1) []
23
24  let create_runtime_functions ctx bt the_module =
25    let add_runtime_func fname returntype arglist =
26      let the_func = Llvm.declare_function fname (Llvm.function_type returntype arglist)
            the_module
27      in Hashtbl.add runtime_functions fname the_func in
28    add_runtime_func "strlen" bt.long_t [|bt.char_p|];
29    add_runtime_func "strcmp" bt.long_t [|bt.char_p; bt.char_p|];
30    add_runtime_func "pow" bt.float_t [|bt.float_t; bt.float_t|] ;
31    add_runtime_func "lrint" bt.int_t [|bt.float_t|] ;
32    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|] ;
33    add_runtime_func "incStack" bt.void_t [||] ;
34    add_runtime_func "getVal" bt.value_p [|bt.var_instance_p; bt.int_t; bt.int_t|] ;
35    add_runtime_func "rg_eq" bt.int_t [|bt.value_p; bt.value_p|] ;
36    add_runtime_func "clone_value" bt.value_p [|bt.value_p;|] ;
37    (* add_runtime_func "freeMe" (Llvm.void_type ctx) [|bt.extend_scope_p;|] ; *)
38    add_runtime_func "getSize" bt.value_p [|bt.var_instance_p;|] ;
39    add_runtime_func "get_variable" bt.var_instance_p [|bt.extend_scope_p; bt.int_t|] ;
40    add_runtime_func "null_init" (Llvm.void_type ctx) [|bt.extend_scope_p|] ;
41    add_runtime_func "debug_print" (Llvm.void_type ctx) [|bt.value_p ; bt.char_p|] ;
42    add_runtime_func "new_string" bt.value_p [|bt.char_p|] ;
43    add_runtime_func "deref_subrange_p" bt.value_p [|bt.subrange_p|];
44    add_runtime_func "debug_print_selection" (Llvm.void_type ctx) [|bt.rhs_selection_p
        |];
45    add_runtime_func "extract_selection" bt.value_p [|bt.value_p; bt.rhs_selection_p; bt
        .int_t; bt.int_t|];
46    add_runtime_func "box_command_line_args" bt.value_p [|bt.int_t; bt.char_p_p|];
47    add_runtime_func "verify_assert" (Llvm.void_type ctx) [|bt.value_p; bt.char_p|];
48    ()
49
50  let translate (globals, functions, externs) =
51
52    (* LLVM Boilerplate *)
53    let context = Llvm.global_context () in
54    let base_module = Llvm.create_module context "Extend" in
55    let base_types = setup_types context in
56
57    (* Declare the runtime functions that we need to call *)
58    create_runtime_functions context base_types base_module ;
59
60    (* Build function_llvalues, which is a StringMap from function name to llvalue.
61     * It includes both functions from external libraries, such as the standard library,
```

```
62      * and functions declared within Extend. *)
63     let declare_library_function fname func accum_map =
64       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
65       let llvm_fname = "extend_" ^ fname in
66       let llvm_fn = Llvm.declare_function llvm_fname llvm_ftype base_module in
67       StringMap.add fname llvm_fn accum_map in
68     let library_functions = StringMap.fold declare_library_function externs StringMap.
          empty in
69     let define_user_function fname func =
70       let llvm_fname = "extend_" ^ fname in
71       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
72       let llvm_fn = Llvm.define_function llvm_fname llvm_ftype base_module in
73       (func, llvm_fn) in
74     let extend_functions = StringMap.mapi define_user_function functions in
75     let function_llvalues = StringMap.fold StringMap.add (StringMap.map snd
          extend_functions) library_functions in
76
77     (* Build the global symbol table *)
78     let (global_symbols, num_globals) = index_map Globals globals in
79     let (extend_fn_numbers, num_extend_fns) = index_map ExtendFunctions extend_functions
           in
80
81     (* Create the global array that will hold each function's array of var_defns. *)
82     let vardefn_ptr = Llvm.const_pointer_null base_types.var_defn_p in
83     let vardefn_array = Array.make (StringMap.cardinal extend_functions) vardefn_ptr in
84     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
85
86     (* Create the pointer to the global scope object *)
87     let global_scope_loc = Llvm.define_global "global_scope_loc" (Llvm.
          const_pointer_null base_types.extend_scope_p) base_module in
88
89     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
90     let main_bod = Llvm.builder_at_end context (Llvm.entry_block main_def) in
91
92     let init_def = Llvm.define_function "initialize_vardefns" (Llvm.function_type (Llvm.
          void_type context) [|||]) base_module in
93     let init_bod = Llvm.builder_at_end context (Llvm.entry_block init_def) in
94
95     let literal_def = Llvm.define_function "initialize_literals" (Llvm.function_type (
          Llvm.void_type context) [|||]) base_module in
96     let literal_bod = Llvm.builder_at_end context (Llvm.entry_block literal_def) in
97
98     (* Create the array of value_ps that will contain the responses to TypeOf(val) *)
99     let null_val_ptr = Llvm.const_pointer_null base_types.value_p in
100    let null_val_array = Array.make (Array.length int_to_type_array) null_val_ptr in
101    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
102    let create_typeof_string i s =
103      let sp = Llvm.build_global_stringptr s "global_typeof_stringptr" literal_bod in
104      let vp = Llvm.build_call (Hashtbl.find runtime_functions "new_string") [|sp|] "
              global_typeof_string" literal_bod in
105      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
106      let _ = Llvm.build_store vp vp_dst literal_bod in
107      () in
108    Array.iteri create_typeof_string int_to_type_array ;
109
110    (* Look these two up once and for all *)
111    (* let deepCopy = Hashtbl.find runtime_functions "deepCopy" in *)
112    (* let freeMe = Hashtbl.find runtime_functions "freeMe" in *)
113    let getVal = Hashtbl.find runtime_functions "getVal" in (*getVal retrieves the value
             of a variable instance for a specific x and y*)
114    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*)
115
116    (* build_formula_function takes a symbol table and an expression, builds the LLVM
             function, and returns the llvalue of the function *)
117    let build_formula_function (varname, formula_idx) symbols formula_expr =
118      let form_decl = Llvm.define_function ("formula_fn_" ^ varname ^ "_num_" ^ (
             string_of_int formula_idx)) base_types.formula_call_t base_module in
119      let builder_at_top = Llvm.builder_at_end context (Llvm.entry_block form_decl) in
120      let local_scope = Llvm.param form_decl 0 in
121      let cell_row = Llvm.param form_decl 1 in
122      let cell_col = Llvm.param form_decl 2 in
123      let global_scope = Llvm.build_load global_scope_loc "global_scope" builder_at_top
             in
124
125      (* Some repeated stuff to avoid cut & paste *)
126      let empty_type = (Llvm.const_int base_types.char_t (value_field_flags_index Empty)
               ) in
127      let number_type = (Llvm.const_int base_types.char_t (value_field_flags_index
               Number)) in
128      let string_type = (Llvm.const_int base_types.char_t (value_field_flags_index
               String)) in
129      let range_type = (Llvm.const_int base_types.char_t (value_field_flags_index Range)
               ) in
130      let make_block blockname =
131        let new_block = Llvm.append_block context blockname form_decl in
132        let new_builder = Llvm.builder_at_end context new_block in
133        (new_block, new_builder) in
134      let store_number value_ptr store_builder number_llvalue =
135        let sp = Llvm.build_struct_gep value_ptr (value_field_index Number) "num_pointer
               " store_builder in
136        let _ = Llvm.build_store number_type (Llvm.build_struct_gep value_ptr (
               value_field_index Flags) "" store_builder) store_builder in
137        ignore (Llvm.build_store number_llvalue sp store_builder) in
138      let store_empty value_ptr store_builder =
139        ignore (Llvm.build_store empty_type (Llvm.build_struct_gep value_ptr (
               value_field_index Flags) "" store_builder) store_builder) in
140
141      let make_truthiness_blocks blockprefix ret_val =
142        let (merge_bb, merge_builder) = make_block (blockprefix ^ "_merge") in
143
144        let (make_true_bb, make_true_builder) = make_block (blockprefix ^ "_true") in
145        let _ = store_number ret_val make_true_builder (Llvm.const_float base_types.
               float_t 1.0) in
```

66

```
146        let _ = Llvm.build_br merge_bb make_true_builder in
147
148        let (make_false_bb, make_false_builder) = make_block (blockprefix ^ "_false") in
149        let _ = store_number ret_val make_false_builder (Llvm.const_float base_types.
               float_t 0.0) in
150        let _ = Llvm.build_br merge_bb make_false_builder in
151
152        let (make_empty_bb, make_empty_builder) = make_block (blockprefix ^ "_empty") in
153        let _ = store_empty ret_val make_empty_builder   in
154        let _ = Llvm.build_br merge_bb make_empty_builder in
155
156        (make_true_bb, make_false_bb, make_empty_bb, merge_builder) in
157
158     let rec build_expr old_builder exp = match exp with
159        LitInt(i) -> let vvv = Llvm.const_float base_types.float_t (float_of_int i) in
160        let ret_val = Llvm.build_malloc base_types.value_t "int_ret_val" old_builder
               in
161        let _ = store_number ret_val old_builder vvv in
162        (ret_val, old_builder)
163      | LitFlt(f) -> let vvv = Llvm.const_float base_types.float_t f in
164        let ret_val = Llvm.build_malloc base_types.value_t "flt_ret_val" old_builder
               in
165        let _ = store_number ret_val old_builder vvv in
166        (ret_val, old_builder)
167      | UnOp(Neg, LitInt(i)) -> build_expr old_builder (LitInt(-i))
168      | UnOp(Neg, LitFlt(f)) -> build_expr old_builder (LitFlt(-.f))
169      | Empty ->
170        let ret_val = Llvm.build_malloc base_types.value_t "empty_ret_val" old_builder
                in
171        let _ = store_empty ret_val old_builder in
172        (ret_val, old_builder)
173      | Debug(e) ->
174        let (ret_val, new_builder) = build_expr old_builder e in
175        let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
               ret_val; Llvm.const_pointer_null base_types.char_p|] "" new_builder in
176        (ret_val, new_builder)
177      | Id(name) ->
178        let create_and_deref_subrange appropriate_scope i =
179          let llvm_var = Llvm.build_call getVar [|appropriate_scope; Llvm.const_int
                 base_types.int_t i|] "llvm_var" old_builder in
180          let base_var_num_rows = (llvm_var => (var_instance_field_index Rows)) "
                 base_var_num_rows" old_builder in
181          let base_var_num_cols = (llvm_var => (var_instance_field_index Cols)) "
                 base_var_num_rows" old_builder in
182          let subrange_ptr = Llvm.build_alloca base_types.subrange_t "subrange_ptr"
                 old_builder in
183          let _ = (llvm_var $> (subrange_ptr, (subrange_field_index BaseRangePtr)))
                 old_builder in
184          let _ = ((Llvm.const_null base_types.int_t) $> (subrange_ptr, (
                 subrange_field_index BaseOffsetRow))) old_builder in
185          let _ = ((Llvm.const_null base_types.int_t) $> (subrange_ptr, (
                 subrange_field_index BaseOffsetCol))) old_builder in
186          let _ = (base_var_num_rows $> (subrange_ptr, (subrange_field_index
                 SubrangeRows))) old_builder in
187          let _ = (base_var_num_cols $> (subrange_ptr, (subrange_field_index
                 SubrangeCols))) old_builder in
```

```
188              (Llvm.build_call (Hashtbl.find runtime_functions "deref_subrange_p") [|
                     subrange_ptr|] "local_id_ret_val" old_builder, old_builder) in
189          (
190            match (try StringMap.find name symbols with Not_found -> raise(LogicError("
                   Something went wrong with your semantic analysis - " ^ name ^ " not found
                   "))) with
191            LocalVariable(i) -> create_and_deref_subrange local_scope i
192            | GlobalVariable(i) -> create_and_deref_subrange global_scope i
193            | FunctionParameter(i) ->
194              let paramarray = (local_scope => (scope_field_type_index FunctionParams))
                     "paramarray" old_builder in
195              let param_addr = Llvm.build_in_bounds_gep paramarray [|Llvm.const_int
                     base_types.int_t i|] "param_addr" old_builder in
196              let param = Llvm.build_load param_addr "param" old_builder in
197              (Llvm.build_call (Hashtbl.find runtime_functions "clone_value") [|param|]
                     "function_param_ret_val" old_builder, old_builder)
198            | ExtendFunction(i) -> raise(LogicError("Something went wrong with your
                     semantic analyis - function " ^ name ^ " used as variable in RHS for " ^
                     varname))
199          )
200      | ReducedTernary(cond_var, true_var, false_var) ->
201        let get_llvm_var name getvar_builder =
202          match (try StringMap.find name symbols with Not_found -> raise(LogicError("
                     Something went wront with your transformation - Reduced Ternary name " ^
                     name ^ " not found"))) with
203            LocalVariable(i) -> Llvm.build_call getVar [|local_scope; Llvm.const_int
                     base_types.int_t i|] "llvm_var" getvar_builder
204            | GlobalVariable(i) -> Llvm.build_call getVar [|global_scope; Llvm.const_int
                      base_types.int_t i|] "llvm_var" getvar_builder
205            | _ -> raise(LogicError("Something went wront with your transformation -
                     Reduced Ternary name " ^ name ^ " not a local or global variable")) in
206
207        let (empty_bb, empty_builder) = make_block "empty" in
208        let (not_empty_bb, not_empty_builder) = make_block "not_empty" in
209        let (truthy_bb, truthy_builder) = make_block "truthy" in
210        let (falsey_bb, falsey_builder) = make_block "falsey" in
211        let (merge_bb, merge_builder) = make_block "merge" in
212
213        let ret_val_addr = Llvm.build_alloca base_types.value_p "tern_ret_val_addr"
                 old_builder in
214        let cond_llvm_var = get_llvm_var cond_var old_builder in
215        let cond_val = Llvm.build_call getVal [|cond_llvm_var; cell_row; cell_col|] "
                 cond_val" old_builder in
216        let cond_val_type = (cond_val => (value_field_index Flags)) "cond_val_type"
                 old_builder in
217        let is_empty = Llvm.build_icmp Llvm.Icmp.Eq empty_type cond_val_type "is_empty
                 " old_builder in
218        let _ = Llvm.build_cond_br is_empty empty_bb not_empty_bb old_builder in
219
220        (* Empty basic block: *)
221        let ret_val_empty = Llvm.build_malloc base_types.value_t "tern_empty"
                 empty_builder in
222        let _ = store_empty ret_val_empty empty_builder in
223        let _ = Llvm.build_store ret_val_empty ret_val_addr empty_builder in
224        let _ = Llvm.build_br merge_bb empty_builder in
225
```

```
226              (* Not empty basic block: *)
227          let the_number = (cond_val => (value_field_index Number)) "the_number"
                 not_empty_builder in
228          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 *)
229          let _ = Llvm.build_cond_br is_not_zero truthy_bb falsey_bb not_empty_builder
                 in
230
231          (* Truthy basic block: *)
232          let truthy_llvm_var = get_llvm_var true_var truthy_builder in
233          let truthy_val = Llvm.build_call getVal [|truthy_llvm_var; cell_row; cell_col
                 |] "truthy_val" truthy_builder in
234          let _ = Llvm.build_store truthy_val ret_val_addr truthy_builder in
235          let _ = Llvm.build_br merge_bb truthy_builder in
236
237          (* Falsey basic block: *)
238          let falsey_llvm_var = get_llvm_var false_var falsey_builder in
239          let falsey_val = Llvm.build_call getVal [|falsey_llvm_var; cell_row; cell_col
                 |] "falsey_val" falsey_builder in
240          let _ = Llvm.build_store falsey_val ret_val_addr falsey_builder in
241          let _ = Llvm.build_br merge_bb falsey_builder in
242
243          let ret_val = Llvm.build_load ret_val_addr "tern_ret_val" merge_builder in
244          (ret_val, merge_builder)
245      | Selection(expr, sel) ->
246          let (expr_val, expr_builder) = build_expr old_builder expr in
247          let build_rhs_index idx_builder = function
248              Abs(e) ->
249              let (idx_expr_val, next_builder) = build_expr idx_builder e in
250              let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
                     next_builder in
251              let _ = (idx_expr_val $> (rhs_idx_ptr, (rhs_index_field_index RhsExprVal))
                     ) next_builder in
252              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
253              (rhs_idx_ptr, next_builder)
254            | Rel(e) ->
255              let (idx_expr_val, next_builder) = build_expr idx_builder e in
256              let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
                     next_builder in
257              let _ = (idx_expr_val $> (rhs_idx_ptr, (rhs_index_field_index RhsExprVal))
                     ) next_builder in
258              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
259              (rhs_idx_ptr, next_builder)
260            | DimensionStart ->
261              let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
                     idx_builder in
262              let _ = ((Llvm.const_pointer_null base_types.value_p) $> (rhs_idx_ptr, (
                     rhs_index_field_index RhsExprVal))) idx_builder in
263              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
```

```
264                    (rhs_idx_ptr, idx_builder)
265                | DimensionEnd ->
266                  let rhs_idx_ptr = Llvm.build_alloca base_types.rhs_index_t "idx_ptr"
                        idx_builder in
267                  let _ = ((Llvm.const_pointer_null base_types.value_p) $> (rhs_idx_ptr, (
                        rhs_index_field_index RhsExprVal))) idx_builder in
268                  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
269                  (rhs_idx_ptr, idx_builder) in
270            let build_rhs_slice slice_builder = function
271                (Some start_idx, Some end_idx) ->
272                  let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
                        slice_builder in
273                  let (start_idx_ptr, next_builder) = build_rhs_index slice_builder
                        start_idx in
274                  let (end_idx_ptr, last_builder) = build_rhs_index next_builder end_idx in
275                  let _ = (start_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
                        RhsSliceStartIdx))) last_builder in
276                  let _ = (end_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
                        RhsSliceEndIdx))) last_builder in
277                  (rhs_slice_ptr,last_builder)
278                | (Some single_idx, None) ->
279                  let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
                        slice_builder in
280                  let (single_idx_ptr, last_builder) = build_rhs_index slice_builder
                        single_idx in
281                  let _ = (single_idx_ptr $> (rhs_slice_ptr, (rhs_slice_field_index
                        RhsSliceStartIdx))) last_builder in
282                  let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
                        rhs_slice_ptr, (rhs_slice_field_index RhsSliceEndIdx))) last_builder in
283                  (rhs_slice_ptr,last_builder)
284                | (None, None) ->
285                  let rhs_slice_ptr = Llvm.build_alloca base_types.rhs_slice_t "slice_ptr"
                        slice_builder in
286                  let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
                        rhs_slice_ptr, (rhs_slice_field_index RhsSliceStartIdx))) slice_builder
                         in
287                  let _ = ((Llvm.const_pointer_null base_types.rhs_index_p) $> (
                        rhs_slice_ptr, (rhs_slice_field_index RhsSliceEndIdx))) slice_builder
                         in
288                  (rhs_slice_ptr,slice_builder)
289                | (None, Some illegal_idx) -> print_endline (string_of_expr exp) ; raise (
                      LogicError("This slice should not be grammatically possible")) in
290            let build_rhs_sel sel_builder = function
291                (Some first_slice, Some second_slice) ->
292                  let rhs_selection_ptr = Llvm.build_alloca base_types.rhs_selection_t "
                        selection_ptr" sel_builder in
293                  let (first_slice_ptr, next_builder) = build_rhs_slice sel_builder
                        first_slice in
294                  let (second_slice_ptr, last_builder) = build_rhs_slice next_builder
                        second_slice in
295                  let _ = (first_slice_ptr $> (rhs_selection_ptr, (rhs_selection_field_index
                         RhsSelSlice1))) last_builder in
296                  let _ = (second_slice_ptr $> (rhs_selection_ptr, (
                        rhs_selection_field_index RhsSelSlice2))) last_builder in
```

```
297                    (rhs_selection_ptr,last_builder)
298                | (Some single_slice, None) ->
299                  let rhs_selection_ptr = Llvm.build_alloca base_types.rhs_selection_t "
                       selection_ptr" sel_builder in
300                  let (single_slice_ptr, last_builder) = build_rhs_slice sel_builder
                       single_slice in
301                  let _ = (single_slice_ptr $> (rhs_selection_ptr, (
                       rhs_selection_field_index RhsSelSlice1))) last_builder in
302                  let _ = ((Llvm.const_pointer_null base_types.rhs_slice_p) $> (
                       rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice2)))
                       last_builder in
303                  (rhs_selection_ptr,last_builder)
304                | (None, None) ->
305                  let rhs_selection_ptr = Llvm.build_alloca base_types.rhs_selection_t "
                       selection_ptr" sel_builder in
306                  let _ = ((Llvm.const_pointer_null base_types.rhs_slice_p) $> (
                       rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice1)))
                       sel_builder in
307                  let _ = ((Llvm.const_pointer_null base_types.rhs_slice_p) $> (
                       rhs_selection_ptr, (rhs_selection_field_index RhsSelSlice2)))
                       sel_builder in
308                  (rhs_selection_ptr,sel_builder)
309                | (None, Some illegal_idx) -> print_endline (string_of_expr exp) ; raise (
                     LogicError("This selection should not be grammatically possible")) in
310              let (selection_ptr, builder_to_end_all_builders) = build_rhs_sel expr_builder
                   sel in
311              (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "
                   debug_print_selection") [|selection_ptr|] "" builder_to_end_all_builders in
                    *)
312              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
313              (* 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 *)
314              (ret_val, builder_to_end_all_builders)
315          | Precedence(a,b) -> let (_, new_builder) = build_expr old_builder a in
                 build_expr new_builder b
316          | LitString(str) ->
317              let initbod_charptr = Llvm.build_global_stringptr str "initbod_charptr"
                   literal_bod in
318              let initbod_val_p = Llvm.build_call (Hashtbl.find runtime_functions "
                   new_string") [|initbod_charptr|] "initbod_val_p" literal_bod in
319              let global_val_p_p = Llvm.define_global "global_litstring_p" (Llvm.
                   const_pointer_null base_types.value_p) base_module in
320              let _ = Llvm.build_store initbod_val_p global_val_p_p literal_bod in
321
322              let local_val_p = Llvm.build_load global_val_p_p "local_value_p" old_builder
                   in
323              let ret_val = Llvm.build_call (Hashtbl.find runtime_functions "clone_value")
                   [|local_val_p|] "ret_val" old_builder in
324              (ret_val, old_builder)
325          | LitRange(rl) ->
326              let num_rows = List.length rl in
327              let num_cols = List.fold_left max 0 (List.map List.length rl) in
328              if num_rows = 1 && num_cols = 1 then build_expr old_builder (List.hd (List.hd
```

```
                 rl))
329          else
330            let global_val_p_p = Llvm.define_global "global_litrange_p" (Llvm.
                 const_pointer_null base_types.value_p) base_module in
331            let initbod_val_p = Llvm.build_malloc base_types.value_t "initbod_val_p"
                 literal_bod in
332            let _ = Llvm.build_store initbod_val_p global_val_p_p literal_bod in
333            let _ = (range_type $> (initbod_val_p, (value_field_index Flags)))
                 literal_bod in
334            let anonymous_subrange_p = Llvm.build_malloc base_types.subrange_t "
                 anonymous_subrange" literal_bod in
335            let _ = (anonymous_subrange_p $> (initbod_val_p, (value_field_index Subrange
                 ))) literal_bod in
336
337            let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
                 subrange_field_index BaseOffsetRow))) literal_bod in
338            let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
                 subrange_field_index BaseOffsetCol))) literal_bod in
339            let _ = ((Llvm.const_int base_types.int_t num_rows) $> (anonymous_subrange_p
                 , (subrange_field_index SubrangeRows))) literal_bod in
340            let _ = ((Llvm.const_int base_types.int_t num_cols) $> (anonymous_subrange_p
                 , (subrange_field_index SubrangeCols))) literal_bod in
341            let anonymous_var_inst_p = Llvm.build_malloc base_types.var_instance_t "
                 anonymous_var_inst" literal_bod in
342            let _ = (anonymous_var_inst_p $> (anonymous_subrange_p, (
                 subrange_field_index BaseRangePtr))) literal_bod in
343
344            let _ = ((Llvm.const_int base_types.int_t num_rows) $> (anonymous_var_inst_p
                 , (var_instance_field_index Rows))) literal_bod in
345            let _ = ((Llvm.const_int base_types.int_t num_cols) $> (anonymous_var_inst_p
                 , (var_instance_field_index Cols))) literal_bod in
346            let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_var_inst_p, (
                 var_instance_field_index NumFormulas))) literal_bod in
347            let _ = ((Llvm.const_pointer_null base_types.resolved_formula_p) $> (
                 anonymous_var_inst_p, (var_instance_field_index Formulas))) literal_bod
                 in
348            let _ = ((Llvm.const_pointer_null base_types.extend_scope_p) $> (
                 anonymous_var_inst_p, (var_instance_field_index Closure))) literal_bod in
349            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
350            let _ = (vals_array $> (anonymous_var_inst_p, (var_instance_field_index
                 Values))) literal_bod in
351            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
352            let _ = (status_array $> (anonymous_var_inst_p, (var_instance_field_index
                 Status))) literal_bod in
353
354            let get_val_p e = let (vp, _) = build_expr literal_bod e in vp in
355            let val_p_list_list = List.map (fun x -> List.map get_val_p x) rl in
356            let cellnums = zero_until (num_rows * num_cols) in
357            let build_empty x =
358              let emptyval = Llvm.build_malloc base_types.value_t ("" ^ (string_of_int x
                 )) literal_bod in
359              let _ = store_empty emptyval literal_bod in
360              let emptydst = Llvm.build_in_bounds_gep vals_array [|Llvm.const_int
                 base_types.int_t x|] "" literal_bod in
```

```
361              let _ = Llvm.build_store emptyval emptydst literal_bod in
362              let statusdst = Llvm.build_in_bounds_gep status_array [|Llvm.const_int
                    base_types.int_t x|] "" literal_bod in
363              let _ = Llvm.build_store (Llvm.const_int base_types.char_t (
                    var_instance_status_flags_index Calculated)) statusdst literal_bod in
364              () in
365          List.iter build_empty cellnums ;
366          let store_val r c realval =
367              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
368              let _ = Llvm.build_store realval realdst literal_bod in
369              () in
370          let store_row r cols = List.iteri (fun c v -> store_val r c v) cols in
371          List.iteri store_row val_p_list_list ;
372          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                    initbod_val_p; Llvm.const_pointer_null base_types.char_p|] "" literal_bod
                    in *)
373
374          let local_val_p = Llvm.build_load global_val_p_p "local_value_p" old_builder
                    in
375          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                    local_val_p; Llvm.const_pointer_null base_types.char_p|] "" old_builder
                    in *)
376          let ret_val = Llvm.build_call (Hashtbl.find runtime_functions "clone_value")
                    [|local_val_p|] "ret_val" old_builder in
377          (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                    ret_val; Llvm.const_pointer_null base_types.char_p|] "" old_builder in *)
378          (ret_val, old_builder)
379      | Call(fn,exl) -> (*TODO: Call needs to be reviewed. Possibly switch call
                arguments to value_p*)
380        let build_one_expr (arg_list, intermediate_builder) e =
381          let (arg_val, next_builder) = build_expr intermediate_builder e in
382          (arg_val :: arg_list, next_builder) in
383        let (reversed_arglist, call_builder) = List.fold_left build_one_expr ([],
                old_builder) exl in
384        let args = Array.of_list (List.rev reversed_arglist) in
385        let result = Llvm.build_call (
386          StringMap.find fn function_llvalues
387          ) args "call_ret_val" call_builder in
388        (result, call_builder)
389      | BinOp(expr1,op,expr2) -> (
390          let (val1, builder1) = build_expr old_builder expr1 in
391          let (val2, int_builder) = build_expr builder1 expr2 in
392          let bit_shift = (Llvm.const_int base_types.char_t 4) in
393          let expr1_type = (val1 => (value_field_index Flags)) "expr1_type"
                    int_builder in
394          let expr2_type = (val2 => (value_field_index Flags)) "expr2_type"
                    int_builder in
395          let expr1_type_shifted = Llvm.build_shl expr1_type bit_shift "
                    expr_1_type_shifted" int_builder in
396          let combined_type = Llvm.build_add expr1_type_shifted expr2_type "
                    combined_type" int_builder in
397          let number_number = Llvm.const_add (Llvm.const_shl number_type bit_shift)
                    number_type in
398          let string_string = Llvm.const_add (Llvm.const_shl string_type bit_shift)
```

73

```
                    string_type in
399        let empty_empty = Llvm.const_add (Llvm.const_shl empty_type bit_shift)
                    empty_type in
400        let range_range = Llvm.const_add (Llvm.const_shl range_type bit_shift)
                    range_type in
401        let build_simple_binop oppp int_builder =
402          (let ret_val = Llvm.build_malloc base_types.value_t "binop_minus_ret_val"
                    int_builder in
403            let _ = Llvm.build_store
404                (
405                  Llvm.const_int
406                  base_types.char_t
407                  (value_field_flags_index Empty)
408                ) (
409                  Llvm.build_struct_gep
410                  ret_val
411                  (value_field_index Flags)
412                  ""
413                  int_builder
414                )
415                int_builder
416            in
417            let bailout = (Llvm.append_block context "" form_decl) in
418            let bbailout = Llvm.builder_at_end context bailout in
419            let (numnum_bb, numnum_builder) = make_block "numnum" in
420            let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                    numnum_builder in
421            let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                    numnum_builder in
422            let numeric_res = oppp numeric_val_1 numeric_val_2 "numeric_res"
                    numnum_builder in
423            let _ = Llvm.build_store
424                numeric_res (
425                  Llvm.build_struct_gep
426                  ret_val
427                  (value_field_index Number)
428                  ""
429                  numnum_builder
430                )
431                numnum_builder in
432            let _ = Llvm.build_store
433                (
434                  Llvm.const_int
435                  base_types.char_t
436                  (value_field_flags_index Number)
437                ) (
438                  Llvm.build_struct_gep
439                  ret_val
440                  (value_field_index Flags)
441                  ""
442                  numnum_builder
443                )
444                numnum_builder in
445            let _ = Llvm.build_br bailout numnum_builder in
446            let _ = Llvm.build_cond_br (Llvm.build_icmp Llvm.Icmp.Eq combined_type
                    number_number "" int_builder) numnum_bb bailout int_builder in
```

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

```
                               number_number "" int_builder) numnum_bb bailout int_builder in
497                   (ret_val, bbailout)
498                 ) in
499            let build_boolean_op numeric_comparator string_comparator int_builder =
500              let ret_val = Llvm.build_malloc base_types.value_t "binop_gt_ret_val"
                      int_builder in
501              let (make_true_bb, make_false_bb, make_empty_bb, merge_builder) =
                      make_truthiness_blocks "binop_eq" ret_val in
502
503              let (numnum_bb, numnum_builder) = make_block "numnum" in
504              let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                      numnum_builder in
505              let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                      numnum_builder in
506              let numeric_greater = Llvm.build_fcmp numeric_comparator numeric_val_1
                      numeric_val_2 "numeric_greater" numnum_builder in
507              let _ = Llvm.build_cond_br numeric_greater make_true_bb make_false_bb
                      numnum_builder in
508
509              let (strstr_bb, strstr_builder) = make_block "strstr" in
510              let str_p_1 = (val1 => (value_field_index String)) "string_one"
                      strstr_builder in
511              let str_p_2 = (val2 => (value_field_index String)) "string_two"
                      strstr_builder in
512              let char_p_1 = (str_p_1 => (string_field_index StringCharPtr)) "char_p_one
                      " strstr_builder in
513              let char_p_2 = (str_p_2 => (string_field_index StringCharPtr)) "char_p_two
                      " strstr_builder in
514              let strcmp_result = Llvm.build_call (Hashtbl.find runtime_functions "
                      strcmp") [|char_p_1; char_p_2|] "strcmp_result" strstr_builder in
515              let string_greater = Llvm.build_icmp string_comparator strcmp_result (Llvm
                      .const_null base_types.long_t) "string_greater" strstr_builder in
516              let _ = Llvm.build_cond_br string_greater make_true_bb make_false_bb
                      strstr_builder in
517
518              let switch_inst = Llvm.build_switch combined_type make_empty_bb 2
                      int_builder in (* Incompatible ===> default to empty *)
519            Llvm.add_case switch_inst number_number numnum_bb;
520            Llvm.add_case switch_inst string_string strstr_bb;
521            (ret_val, merge_builder) in
522          match op with
523            Minus -> build_simple_binop Llvm.build_fsub int_builder
524          | Plus ->
525              let result = Llvm.build_malloc base_types.value_t "" int_builder
526              and stradd = (Llvm.append_block context "" form_decl)
527              and numadd = (Llvm.append_block context "" form_decl)
528              and bailout = (Llvm.append_block context "" form_decl)
529              and numorstrorother = (Llvm.append_block context "" form_decl)
530              and strorother = (Llvm.append_block context "" form_decl)
531              in
532              let bstradd = Llvm.builder_at_end context stradd
533              and bnumadd = Llvm.builder_at_end context numadd
534              and bnumorstrorother = Llvm.builder_at_end context numorstrorother
535              and bstrorother = Llvm.builder_at_end context strorother
536              and bbailout = Llvm.builder_at_end context bailout
537              and _ = Llvm.build_store (Llvm.const_int base_types.char_t (
```

76

```
                     value_field_flags_index Empty)) (Llvm.build_struct_gep result (
                     value_field_index Flags) "" int_builder) int_builder
538              in
539              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
540              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
541              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
542              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
543              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
544              and str1 = Llvm.build_load (Llvm.build_struct_gep val1 (
                     value_field_index String) "" bstradd) "" bstradd
545              and str2 = Llvm.build_load (Llvm.build_struct_gep val2 (
                     value_field_index String) "" bstradd) "" bstradd
546              and newstr = (Llvm.build_malloc base_types.string_t "" bstradd) in
547              let len1 = Llvm.build_load (Llvm.build_struct_gep str1 (
                     string_field_index StringLen) "" bstradd) "" bstradd
548              and len2 = Llvm.build_load (Llvm.build_struct_gep str2 (
                     string_field_index StringLen) "" bstradd) "" bstradd
549              and p1 = Llvm.build_load (Llvm.build_struct_gep str1 (string_field_index
                      StringCharPtr) "" bstradd) "" bstradd
550              and p2 = Llvm.build_load (Llvm.build_struct_gep str2 (string_field_index
                      StringCharPtr) "" bstradd) "" bstradd
551              and dst_char_ptr_ptr = (Llvm.build_struct_gep newstr (string_field_index
                      StringCharPtr) "" bstradd)
552              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
553              and _ = Llvm.build_store newstr (Llvm.build_struct_gep result (
                     value_field_index String) "" bstradd) bstradd in
554              let fullLen = Llvm.build_nsw_add (Llvm.build_nsw_add len1 len2 ""
                     bstradd) (Llvm.const_int base_types.long_t 1) "" bstradd
555              and extra_byte2 = (Llvm.build_add len2 (Llvm.const_int base_types.long_t
                      1) "" bstradd) in
556              let dst_char = Llvm.build_array_malloc base_types.char_t (Llvm.
                     build_trunc fullLen base_types.int_t "" bstradd) "" bstradd in
557              let dst_char2 = Llvm.build_in_bounds_gep dst_char [|len1|] "" bstradd in
558              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
559              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
560                  and _ = Llvm.build_store dst_char dst_char_ptr_ptr bstradd
561                  in
562                  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
563                  in
564                  let _ = Llvm.build_cond_br isnumorstring numorstrorother bailout
                          int_builder
565                  and _ = Llvm.build_cond_br isnumber numadd strorother bnumorstrorother
566                  and _ = Llvm.build_cond_br isstring stradd bailout bstrorother
567                  and _ = Llvm.build_br bailout bstradd
568                  and _ = Llvm.build_br bailout bnumadd
569                  in
570                  (result, bbailout)
571            | Times −> build_simple_binop Llvm.build_fmul int_builder
572            | Eq −>
573              (* let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print")
                      [|val1; Llvm.build_global_stringptr "Eq operator − value 1" ""
                      old_builder|] "" int_builder in
574              let _ = Llvm.build_call (Hashtbl.find runtime_functions "debug_print") [|
                      val2; Llvm.build_global_stringptr "Eq operator − value 2" ""
                      old_builder|] "" int_builder in *)
575              let ret_val = Llvm.build_malloc base_types.value_t "binop_eq_ret_val"
                      int_builder in
576              let (make_true_bb, make_false_bb, _, merge_builder) =
                      make_truthiness_blocks "binop_eq" ret_val in
577
578              let (numnum_bb, numnum_builder) = make_block "numnum" in
579              let numeric_val_1 = (val1 => (value_field_index Number)) "number_one"
                      numnum_builder in
580              let numeric_val_2 = (val2 => (value_field_index Number)) "number_two"
                      numnum_builder in
581              let numeric_equality = Llvm.build_fcmp Llvm.Fcmp.Oeq numeric_val_1
                      numeric_val_2 "numeric_equality" numnum_builder in
582              let _ = Llvm.build_cond_br numeric_equality make_true_bb make_false_bb
                      numnum_builder in
583
584              let (strstr_bb, strstr_builder) = make_block "strstr" in
585              let str_p_1 = (val1 => (value_field_index String)) "string_one"
                      strstr_builder in
586              let str_p_2 = (val2 => (value_field_index String)) "string_two"
                      strstr_builder in
587              let char_p_1 = (str_p_1 => (string_field_index StringCharPtr)) "char_p_one
                      " strstr_builder in
588              let char_p_2 = (str_p_2 => (string_field_index StringCharPtr)) "char_p_two
                      " strstr_builder in
589              let strcmp_result = Llvm.build_call (Hashtbl.find runtime_functions "
                      strcmp") [|char_p_1; char_p_2|] "strcmp_result" strstr_builder in
590              let string_equality = Llvm.build_icmp Llvm.Icmp.Eq strcmp_result (Llvm.
                      const_null base_types.long_t) "string_equality" strstr_builder in
591              let _ = Llvm.build_cond_br string_equality make_true_bb make_false_bb
                      strstr_builder in
592
```

```
593              let (rngrng_bb, rngrng_builder) = make_block "rngrng" in
594              (* TODO: Make this case work *)
595              let eqt = Llvm.build_is_not_null (Llvm.build_call (Hashtbl.find
                     runtime_functions "rg_eq") [|val1; val2|] "" rngrng_builder) ""
                     rngrng_builder in
596              let _ = Llvm.build_cond_br eqt make_true_bb make_false_bb rngrng_builder
                     in
597
598              let switch_inst = Llvm.build_switch combined_type make_false_bb 4
                     int_builder in (* Incompatible ===> default to false *)
599            Llvm.add_case switch_inst number_number numnum_bb;
600            Llvm.add_case switch_inst string_string strstr_bb;
601            Llvm.add_case switch_inst range_range rngrng_bb;
602            Llvm.add_case switch_inst empty_empty make_true_bb; (* Nothing to check in
                    this case, just return true *)
603            (ret_val, merge_builder)
604          | Gt -> build_boolean_op Llvm.Fcmp.Ogt Llvm.Icmp.Sgt int_builder
605          | GtEq -> build_boolean_op Llvm.Fcmp.Oge Llvm.Icmp.Sge int_builder
606          | Lt -> build_boolean_op Llvm.Fcmp.Olt Llvm.Icmp.Slt int_builder
607          | LtEq -> build_boolean_op Llvm.Fcmp.Ole Llvm.Icmp.Sle int_builder
608          | 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))
609          | Divide-> build_simple_binop Llvm.build_fdiv int_builder
610          | Mod-> build_simple_binop Llvm.build_frem int_builder
611          | Pow-> (
612            let powcall numeric_val_1 numeric_val_2 valname b =
613              Llvm.build_call (Hashtbl.find runtime_functions "pow") [|numeric_val_1;
                     numeric_val_2|] "" b in
614            build_simple_binop powcall int_builder)
615          | LShift-> build_simple_int_binop Llvm.build_shl int_builder
616          | RShift-> build_simple_int_binop Llvm.build_lshr int_builder
617          | BitOr-> build_simple_int_binop Llvm.build_or int_builder
618          | BitAnd-> build_simple_int_binop Llvm.build_and int_builder
619          | BitXor-> build_simple_int_binop Llvm.build_xor int_builder
620          )
621      | UnOp(SizeOf,expr) ->
622        let ret_val = Llvm.build_malloc base_types.value_t "unop_size_ret_val"
                old_builder in
623
624        (* TODO: We actually have to keep track of these anonymous objects somewhere
                so we can free them *)
625        let _ = (range_type $> (ret_val, (value_field_index Flags))) old_builder in
626        let anonymous_subrange_p = Llvm.build_malloc base_types.subrange_t "
                anonymous_subrange" old_builder in
627        let _ = (anonymous_subrange_p $> (ret_val, (value_field_index Subrange)))
                old_builder in
628
629        let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
                subrange_field_index BaseOffsetRow))) old_builder in
630        let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_subrange_p, (
                subrange_field_index BaseOffsetCol))) old_builder in
631        let _ = ((Llvm.const_int base_types.int_t 1) $> (anonymous_subrange_p, (
                subrange_field_index SubrangeRows))) old_builder in
632        let _ = ((Llvm.const_int base_types.int_t 2) $> (anonymous_subrange_p, (
                subrange_field_index SubrangeCols))) old_builder in
```

```
633    let anonymous_var_inst_p = Llvm.build_malloc base_types.var_instance_t "
           anonymous_var_inst" old_builder in
634    let _ = (anonymous_var_inst_p $> (anonymous_subrange_p, (subrange_field_index
           BaseRangePtr))) old_builder in
635
636    let _ = ((Llvm.const_int base_types.int_t 1) $> (anonymous_var_inst_p, (
           var_instance_field_index Rows))) old_builder in
637    let _ = ((Llvm.const_int base_types.int_t 2) $> (anonymous_var_inst_p, (
           var_instance_field_index Cols))) old_builder in
638    let _ = ((Llvm.const_int base_types.int_t 0) $> (anonymous_var_inst_p, (
           var_instance_field_index NumFormulas))) old_builder in
639    let _ = ((Llvm.const_pointer_null base_types.resolved_formula_p) $> (
           anonymous_var_inst_p, (var_instance_field_index Formulas))) old_builder in
640    let _ = ((Llvm.const_pointer_null base_types.extend_scope_p) $> (
           anonymous_var_inst_p, (var_instance_field_index Closure))) old_builder in
641    let num_rows_val = Llvm.build_malloc base_types.value_t "num_rows_val"
           old_builder in
642    let num_cols_val = Llvm.build_malloc base_types.value_t "num_cols_val"
           old_builder in
643    let vals_array = Llvm.build_array_malloc base_types.value_p (Llvm.const_int
           base_types.int_t 2) "vals_array" old_builder in
644    let _ = (vals_array $> (anonymous_var_inst_p, (var_instance_field_index Values
           ))) old_builder in
645    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
646    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
647    let status_array = Llvm.build_array_malloc base_types.char_t (Llvm.const_int
           base_types.int_t 2) "status_array" old_builder in
648    let _ = (status_array $> (anonymous_var_inst_p, (var_instance_field_index
           Status))) old_builder in
649    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
650    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
651
652    let (expr_val, expr_builder) = build_expr old_builder expr in
653    let val_flags = (expr_val => (value_field_index Flags)) "val_flags"
           expr_builder in
654    let is_subrange = Llvm.build_icmp Llvm.Icmp.Eq val_flags range_type "
           is_subrange" expr_builder in
655
656    let (merge_bb, merge_builder) = make_block "merge" in
657
658    let (primitive_bb, primitive_builder) = make_block "primitive" in
659    let _ = store_number num_rows_val primitive_builder (Llvm.const_float
           base_types.float_t 1.0) in
660    let _ = store_number num_cols_val primitive_builder (Llvm.const_float
           base_types.float_t 1.0) in
661    let _ = Llvm.build_br merge_bb primitive_builder in
662
663    let (subrange_bb, subrange_builder) = make_block "subrange" in
```

```
664        let subrange_ptr = (expr_val => (value_field_index Subrange)) "subrange_ptr"
               subrange_builder in
665        let rows_as_int = (subrange_ptr => (subrange_field_index SubrangeRows)) "
               rows_as_int" subrange_builder in
666        let cols_as_int = (subrange_ptr => (subrange_field_index SubrangeCols)) "
               cols_as_int" subrange_builder in
667        let rows_as_float = Llvm.build_sitofp rows_as_int base_types.float_t "
               rows_as_float" subrange_builder in
668        let cols_as_float = Llvm.build_sitofp cols_as_int base_types.float_t "
               cols_as_float" subrange_builder in
669        let _ = store_number num_rows_val subrange_builder rows_as_float in
670        let _ = store_number num_cols_val subrange_builder cols_as_float in
671        let _ = Llvm.build_br merge_bb subrange_builder in
672
673        let _ = Llvm.build_cond_br is_subrange subrange_bb primitive_bb expr_builder
               in
674        (ret_val, merge_builder)
675      | UnOp(Truthy, expr) ->
676        let ret_val = Llvm.build_malloc base_types.value_t "unop_truthy_ret_val"
               old_builder in
677        let (expr_val, expr_builder) = build_expr old_builder expr in
678
679        let (truthy_bb, falsey_bb, empty_bb, merge_builder) = make_truthiness_blocks "
               unop_truthy" ret_val in
680
681        let expr_flags = (expr_val => (value_field_index Flags)) "expr_flags"
               expr_builder in
682        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
683        let is_empty = Llvm.build_zext is_empty_bool base_types.char_t "is_empty"
               expr_builder in
684        let is_empty_two = Llvm.build_shl is_empty (Llvm.const_int base_types.char_t
               1) "is_empty_two" expr_builder in
685        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
686        let the_number = (expr_val => (value_field_index Number)) "the_number"
               expr_builder in
687        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
688        let is_numeric_zero_bool = Llvm.build_and is_zero is_number "
               is_numeric_zero_bool" expr_builder in
689        let is_numeric_zero = Llvm.build_zext is_numeric_zero_bool base_types.char_t "
               is_numeric_zero" expr_builder in
690        let switch_num = Llvm.build_add is_empty_two is_numeric_zero "switch_num"
               expr_builder in
691        let switch_inst = Llvm.build_switch switch_num empty_bb 2 expr_builder in
692        Llvm.add_case switch_inst (Llvm.const_int base_types.char_t 0) truthy_bb; (*
               empty << 1 + is_zero == 0 ===> truthy *)
693        Llvm.add_case switch_inst (Llvm.const_int base_types.char_t 1) falsey_bb; (*
               empty << 1 + is_zero == 1 ===> falsey *)
694        (ret_val, merge_builder)
695      | UnOp(LogNot, expr) ->
696        let (truth_val, truth_builder) = build_expr old_builder (UnOp(Truthy, expr))
               in
```

```
697           let the_number = (truth_val => (value_field_index Number)) "the_number"
                  truth_builder in
698         let not_the_number = Llvm.build_fsub (Llvm.const_float base_types.float_t 1.0)
                   the_number "not_the_number" truth_builder in
699         let sp = Llvm.build_struct_gep truth_val (value_field_index Number) "
                  num_pointer" truth_builder in
700         let _ = Llvm.build_store not_the_number sp truth_builder in
701         (truth_val, truth_builder)
702      | UnOp(Neg, expr) ->
703         let ret_val = Llvm.build_malloc base_types.value_t "unop_truthy_ret_val"
                  old_builder in
704         let _ = store_empty ret_val old_builder in
705         let (expr_val, expr_builder) = build_expr old_builder expr in
706         let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
                  expr_builder in
707         let is_number = Llvm.build_icmp Llvm.Icmp.Eq expr_type number_type "is_number"
                   expr_builder in
708         let (finish_bb, finish_builder) = make_block "finish" in
709
710         let (number_bb, number_builder) = make_block "number" in
711         let the_number = (expr_val => (value_field_index Number)) "the_number"
                  number_builder in
712         let minus_the_number = Llvm.build_fneg the_number "minus_the_number"
                  number_builder in
713         let _ = store_number ret_val number_builder minus_the_number in
714         let _ = Llvm.build_br finish_bb number_builder in
715
716         let _ = Llvm.build_cond_br is_number number_bb finish_bb expr_builder in
717         (ret_val, finish_builder)
718      | UnOp(BitNot, expr) ->
719         let ret_val = Llvm.build_malloc base_types.value_t "unop_truthy_ret_val"
                  old_builder in
720         let (expr_val, expr_builder) = build_expr old_builder expr in
721
722         let (numnum_bb, numnum_builder) = make_block "numnum" in
723         let (make_empty_bb, make_empty_builder) = make_block ("" ^ "_empty") in
724         let (finish_bb, finish_builder) = make_block "finish" in
725
726         let _ = store_empty ret_val make_empty_builder  in
727         let _ = Llvm.build_br finish_bb make_empty_builder in
728
729         let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
                  expr_builder in
730         let is_number = Llvm.build_icmp Llvm.Icmp.Eq expr_type number_type "is_number"
                   expr_builder in
731         let _ = Llvm.build_cond_br is_number numnum_bb make_empty_bb expr_builder in
732
733         let expr_num = Llvm.build_call (Hashtbl.find runtime_functions "lrint") [|((
                  expr_val => (value_field_index Number)) "expr_type" numnum_builder)|] ""
                  numnum_builder in
734         let _ = store_number ret_val numnum_builder (Llvm.build_sitofp (Llvm.build_not
                   expr_num "" numnum_builder) base_types.float_t "" numnum_builder) in
735         let _ = Llvm.build_br finish_bb numnum_builder in
736
737         (ret_val, finish_builder)
738      | UnOp(TypeOf, expr) ->
```

82

```
739          let (expr_val, expr_builder) = build_expr old_builder expr in
740          let expr_type = (expr_val => (value_field_index Flags)) "expr_type"
                 expr_builder in
741          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
742          let vp_to_clone = Llvm.build_load vp_to_clone_loc "vp_to_clone" expr_builder
                 in
743          let ret_val = Llvm.build_call (Hashtbl.find runtime_functions "clone_value")
                 [|vp_to_clone|] "typeof_ret_val" expr_builder in
744          (ret_val, expr_builder)
745        | UnOp(Row, _) ->
746          let row_as_int = cell_row in
747          let row_as_float = Llvm.build_sitofp row_as_int base_types.float_t "
                 row_as_float" old_builder in
748          let ret_val = Llvm.build_malloc base_types.value_t "ret_val" old_builder in
749          let _ = store_number ret_val old_builder row_as_float in
750          (ret_val, old_builder)
751        | UnOp(Column, _) ->
752          let col_as_int = cell_col in
753          let col_as_float = Llvm.build_sitofp col_as_int base_types.float_t "
                 col_as_float" old_builder in
754          let ret_val = Llvm.build_malloc base_types.value_t "ret_val" old_builder in
755          let _ = store_number ret_val old_builder col_as_float in
756          (ret_val, old_builder)
757        | Switch(_,_,_) | Ternary(_,_,_) -> raise(TransformedAway("These expressions
                 should have been transformed away")) in
758        (* | unknown_expr -> print_endline (string_of_expr unknown_expr);raise
                 NotImplemented in *)
759      let (ret_value_p, final_builder) = build_expr builder_at_top formula_expr in
760      let _ = Llvm.build_ret ret_value_p final_builder in
761      form_decl in
762
763    (*build formula creates a formula declaration in a separate method from the function
            it belongs to*)
764    let build_formula (varname, idx) formula_array element symbols =
765      let storage_addr = Llvm.build_in_bounds_gep formula_array [|Llvm.const_int
             base_types.int_t idx|] "" init_bod in
766      let getStarts = function (* Not really just for starts *)
767          Abs(LitInt(1)) | Abs(LitInt(0)) | DimensionStart | DimensionEnd -> (1, -1)
768        | Abs(Id(s)) ->
769          (match StringMap.find s symbols with
770            LocalVariable(i) | GlobalVariable(i) -> (0, i)
771           | _ -> 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.")))
772        | _ -> 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
773      let getEnds = function
774          Some x -> let (b, c) = getStarts x in (b, c, 0)
775        | None -> (0, -1, 1) in
776      let (fromStartRow, rowStartVarnum) = getStarts element.formula_row_start in
777      let (fromStartCol, colStartVarnum) = getStarts element.formula_col_start in
778      let (toEndRow, rowEndVarnum, isSingleRow) = getEnds element.formula_row_end in
779      let (toEndCol, colEndVarnum, isSingleCol) = getEnds element.formula_col_end in
```

83

```
780
781      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
782      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
783      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
784      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
785      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
786
787      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
788      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
789      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
790      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
791      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
792
793      let form_decl = build_formula_function (varname, idx) symbols element.formula_expr
              in
794      let _ = Llvm.build_store form_decl (Llvm.build_struct_gep storage_addr (
             formula_field_index FormulaCall) "" init_bod) init_bod in
795      () in
796
797    (* Builds a var_defn struct for each variable *)
798    let build_var_defn defn varname va symbols =
799      let numForm = List.length va.var_formulas in
800      let formulas = Llvm.build_array_malloc base_types.formula_t (Llvm.const_int
             base_types.int_t numForm) "" init_bod in
801      (*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*)
802      let getDefn = function
803        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.")))
804      | DimOneByOne -> 1 in
805      let _ = (match va.var_rows with
806          DimOneByOne -> Llvm.build_store (Llvm.const_int base_types.char_t 1) (Llvm.
               build_struct_gep defn (var_defn_field_index OneByOne) "" init_bod)
```

84

```
                                 init_bod
807         | DimId(a) -> (
808             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 ();
809             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 ();
810             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
811         )
812       ) in
813     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
814       and _ = Llvm.build_store formulas (Llvm.build_struct_gep defn (
            var_defn_field_index Formulas) "" init_bod) init_bod
815       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
816     List.iteri (fun idx elem -> build_formula (varname, idx) formulas elem symbols) va
            .var_formulas in
817
818   (* Creates a scope object and inserts the necessary instructions into main to
          populate the var_defns, and
819    * into the function specified by builder to populate the scope object. *)
820   let build_scope_obj
821       fname (* The function name, or "globals" *)
822       symbols (* The symbols to use when creating the functions *)
823       vars (* The variables to build definitions and formula-functions for *)
824       static_location_ptr (* The copy of the global pointer used in main *)
825       var_defns_loc (* The copy of the global pointer used in the local function *)
826       num_params (* How many parameters the function takes *)
827       builder (* The LLVM builder for the local function *)
828     =
829     let cardinal = Llvm.const_int base_types.int_t (StringMap.cardinal vars) in
830     let build_var_defns =
831       let static_var_defns = Llvm.build_array_malloc base_types.var_defn_t cardinal (
              fname ^ "_static_var_defns") init_bod in
832       let _ = Llvm.build_store static_var_defns static_location_ptr init_bod in
833       let add_variable varname va (sm, count) =
834         let fullname = fname ^ "_" ^ varname in
835         let defn = (Llvm.build_in_bounds_gep static_var_defns [|Llvm.const_int
                base_types.int_t count|] (fullname ^ "_defn") init_bod) in
836         let _ = build_var_defn defn fullname va symbols in
837         (StringMap.add varname count sm, count + 1) in
838       ignore (StringMap.fold add_variable vars (StringMap.empty, 0)) in
839
840     let var_defns = Llvm.build_load var_defns_loc (fname ^ "_global_defn_ptr_loc")
            builder in
841     let var_insts = Llvm.build_array_malloc base_types.var_instance_p cardinal "
            var_insts" builder in
842     let scope_obj = Llvm.build_malloc base_types.extend_scope_t "scope_obj" builder in
843
844     (*Store variable definition and instance*)
845     let _ = Llvm.build_store var_defns (Llvm.build_struct_gep scope_obj (
            scope_field_type_index VarDefn) "" builder) builder in
```

```
846       let _ = Llvm.build_store var_insts (Llvm.build_struct_gep scope_obj (
              scope_field_type_index VarInst) "" builder) builder in
847       let _ = Llvm.build_store cardinal (Llvm.build_struct_gep scope_obj (
              scope_field_type_index VarNum) "" builder) builder in
848       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
849       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
850       let _ = Llvm.build_store paramarray (Llvm.build_struct_gep scope_obj (
              scope_field_type_index FunctionParams) "" builder) builder in
851       let copy_fn_arg i =
852         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
853         ignore (Llvm.build_store (Llvm.param (StringMap.find fname function_llvalues) i)
                 param_addr builder) in
854     List.iter copy_fn_arg (zero_until num_params);
855       let _ = Llvm.build_call (Hashtbl.find runtime_functions "null_init") [|scope_obj|]
               "" builder in
856     build_var_defns ; scope_obj in
857   (* End of build_scope_obj *)
858
859   let build_function fname (fn_def, fn_llvalue) =
860     (* Build the symbol table for this function *)
861     let symbols = create_symbol_table global_symbols fn_def in
862     let fn_idx = match StringMap.find fname extend_fn_numbers with ExtendFunction(i)
           -> i | _ -> raise(LogicError(fname ^ " not in function table")) in
863     let builder = Llvm.builder_at_end context (Llvm.entry_block fn_llvalue) in
864     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
865     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
866     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
867     let get_special_val special_name = function
868         Id(s) -> (match (try StringMap.find s symbols with Not_found -> raise(
              LogicError("Something went wrong with your semantic analysis — " ^ s ^ "
              not found"))) with
869           LocalVariable(i) ->
870           let llvm_var = Llvm.build_call getVar [|scope_obj; Llvm.const_int
                base_types.int_t i|] (special_name ^ "_var") builder in
871           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
872         | _ -> raise(TransformedAway("Error in " ^ fname ^ ": The " ^ special_name ^
                " value should always have been transformed into a local variable")))
873       | _ -> raise(TransformedAway("Error in " ^ fname ^ ": The " ^ special_name ^ "
            value should always have been transformed into a local variable")) in
874     let assert_val = get_special_val "assert" (List.hd fn_def.func_asserts) in
875     let _ = Llvm.build_call (Hashtbl.find runtime_functions "verify_assert") [|
          assert_val; Llvm.build_global_stringptr fname "" builder|] "" builder in
876     let ret_val = get_special_val "return" (snd fn_def.func_ret_val) in
877     let _ = Llvm.build_ret ret_val builder in () in
878   (* End of build_function *)
```

```
879
880     (* Build the global scope object *)
881     let vardefn_p_p = Llvm.build_alloca base_types.var_defn_p "v_p_p" init_bod in
882     let global_scope_obj = build_scope_obj "globals" global_symbols globals vardefn_p_p
            vardefn_p_p 0 init_bod in
883     let _ = Llvm.build_call (Hashtbl.find runtime_functions "incStack") [||] "" init_bod
            in
884     let _ = Llvm.build_store global_scope_obj global_scope_loc init_bod in
885
886     (*iterates over function definitions*)
887     StringMap.iter build_function extend_functions ;
888
889     (* Define the LLVM entry point for the program *)
890     let extend_entry_point = StringMap.find "main" function_llvalues in
891     let _ = Llvm.build_ret_void init_bod in
892     let _ = Llvm.build_ret_void literal_bod in
893     let _ = Llvm.build_call init_def [||] "" main_bod in
894     let _ = Llvm.build_call literal_def [||] "" main_bod in
895     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
896     let _ = Llvm.build_call extend_entry_point [|cmd_line_args|] "" main_bod in
897     let _ = Llvm.build_ret (Llvm.const_int base_types.int_t 0) main_bod in
898
899     base_module
900
901 let build_this ast_mapped =
902     let modu = (translate ast_mapped) in
903     let _ = Llvm_analysis.assert_valid_module modu in
904     modu
```

7.8   linker.ml

```
 1  module StringSet = Set.Make(String)
 2  let link xtndOut ast compiler outputFile =
 3    let tmpFilenameLL = Filename.temp_file "" ".ll"
 4    and tmpFilenameC = Filename.temp_file "" ".o"
 5    and getExterns (_,_,extern) =
 6      StringSet.elements
 7        (Ast.StringMap.fold
 8          (fun key value store -> StringSet.add value.Ast.extern_fn_libname store)
 9          extern
10          StringSet.empty) in
11    let tmpChan = open_out tmpFilenameLL in
12    output_string tmpChan xtndOut; close_out tmpChan;
13    let call1 = (String.concat " " ("llc-3.8" :: "-filetype=obj" :: tmpFilenameLL :: "-o
        " :: tmpFilenameC :: []))
14    and call2 = (String.concat " " (compiler :: "-o" :: outputFile :: tmpFilenameC :: (
        getExterns ast) @ ["runtime.o"])) ^ " -lm" in
15    let resc1 = Sys.command call1 in
16    if resc1 == 0 then (
17      Sys.remove tmpFilenameLL;
18      let resc2 = Sys.command call2 in
19        Sys.remove tmpFilenameC;
20        if resc2 == 0 then () else raise Not_found
```

```
21        )
22      else (Sys.remove tmpFilenameC;raise Not_found)
```

7.9   main.ml

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

7.10   lib.c

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  #include<string.h>
5  #include<stdbool.h>
6  /* #include <sys/time.h> */
7  #include <time.h>
8  #include "runtime.h"
```

```
 9
10  #define MAX_FILES 255
11  FILE *open_files[1 + MAX_FILES] = {NULL};
12  int open_num_files = 0;
13
14  value_p extend_print(value_p whatever, value_p text) {
15    if(!assertSingleString(text)) return new_val();
16    if(!assertText(text)) return new_val();
17    printf("%s", text->str->text);
18    return new_val();
19  }
20
21  value_p extend_printv(value_p whatever, value_p text) {
22    printf("%s", text->str->text);
23    return new_val();
24  }
25
26  value_p extend_printd(value_p whatever, value_p text) {
27    printf("%f\n", text->numericVal);
28    value_p result = malloc(sizeof(struct value_t));
29    return result;
30  }
31
32  value_p extend_to_string(value_p val) {
33      if(assertSingleNumber(val)) {
34        double possible_num = val->numericVal;
35        int rounded_int = (int) lrint(possible_num);
36        char *converted_str;
37        if (fabs(possible_num − rounded_int) < FLOAT_CUTOFF) {
38          int size = snprintf(NULL, 0, "%d", rounded_int);
39          converted_str = malloc(size + 1);
40          sprintf(converted_str, "%d", rounded_int);
41        } else {
42          int size = snprintf(NULL, 0, "%f", possible_num);
43          converted_str = malloc(size + 1);
44          sprintf(converted_str, "%f", possible_num);
45        }
46        value_p result = new_string(converted_str);
47        return result;
48      }
49      else if(assertSingleString(val)) return val;
50      else if(val->flags == FLAG_EMPTY) {
51        value_p _new = new_val();
52        setString(_new, "empty", 5);
53        return _new;
54      }
55      else if(val->flags == FLAG_SUBRANGE) {
56        int i,j,len;
57        value_p value;
58        char *result, *res;
59        len = 0;
60        subrange_p sr = val->subrange;
61        value_p *strs = malloc(sizeof(value_p) * sr->subrange_num_cols * sr->
              subrange_num_rows);
62        for(i = 0; i < sr->subrange_num_rows; i++) {
63          for(j = 0; j < sr->subrange_num_cols; j++) {
```

```
64          value = extend_to_string(getValSR(sr, i, j));
65          //debug_print(value, "");
66          strs[i * sr->subrange_num_cols + j] = value;
67          len += value->str->length;
68        }
69      }
70      len += sr->subrange_num_rows * sr->subrange_num_cols + 1 /*closing paren*/;
71      res = result = malloc(len + 1/*terminal character*/);
72      *result = '{';
73      result++;
74      for(i = 0; i < sr->subrange_num_rows; i++) {
75        for(j = 0; j < sr->subrange_num_cols; j++) {
76          memcpy(result,strs[i * sr->subrange_num_cols + j]->str->text, strs[i * sr->
                subrange_num_cols + j]->str->length);
77          result += strs[i * sr->subrange_num_cols + j]->str->length;
78          if(j != sr->subrange_num_cols - 1) {
79            *result = ',';
80            result++;
81          }
82        }
83        if(i != sr->subrange_num_rows - 1) {
84          *result = ';';
85          result++;
86        }
87      }
88      *result = '}';
89      value_p ret_val = new_val();
90      setString(ret_val, res, len);
91      return ret_val;
92    } else {
93      __builtin_unreachable();
94    }
95    // If the struct does not hold a string or number, return empty?
96    return new_val();
97  }
98
99  #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);}
100 EXPOSE_MATH_FUNC(sin)
101 EXPOSE_MATH_FUNC(cos)
102 EXPOSE_MATH_FUNC(tan)
103 EXPOSE_MATH_FUNC(acos)
104 EXPOSE_MATH_FUNC(asin)
105 EXPOSE_MATH_FUNC(atan)
106 EXPOSE_MATH_FUNC(sinh)
107 EXPOSE_MATH_FUNC(cosh)
108 EXPOSE_MATH_FUNC(tanh)
109 EXPOSE_MATH_FUNC(exp)
110 EXPOSE_MATH_FUNC(log)
111 EXPOSE_MATH_FUNC(log10)
112 EXPOSE_MATH_FUNC(sqrt)
113 EXPOSE_MATH_FUNC(ceil)
114 EXPOSE_MATH_FUNC(fabs)
115 EXPOSE_MATH_FUNC(floor)
116
117 value_p extend_get_stdin() {
```

```
118    if (open_num_files + 1 > MAX_FILES) {
119      return new_val();
120    } else {
121      open_num_files++;
122      open_files[open_num_files] = stdin;
123      return new_number((double) open_num_files);
124    }
125  }
126
127  value_p extend_get_stdout() {
128    if (open_num_files + 1 > MAX_FILES) {
129      return new_val();
130    } else {
131      open_num_files++;
132      open_files[open_num_files] = stdout;
133      return new_number((double) open_num_files);
134    }
135  }
136
137  value_p extend_get_stderr() {
138    if (open_num_files + 1 > MAX_FILES) {
139      return new_val();
140    } else {
141      open_num_files++;
142      open_files[open_num_files] = stderr;
143      return new_number((double) open_num_files);
144    }
145  }
146
147  value_p extend_open(value_p filename, value_p mode){
148    FILE *val;
149    if (   !assertSingleString(filename)
150        || !assertSingleString(mode)
151        || open_num_files + 1 > MAX_FILES) {
152          return new_val();
153    }
154    val = fopen(filename->str->text, mode->str->text);
155    if(val == NULL) return new_val();
156    open_num_files++;
157    open_files[open_num_files] = val;
158    return new_number((double) open_num_files);
159  }
160
161  value_p extend_close(value_p file_handle) {
162    if(!assertSingleNumber(file_handle)) {
163      // Per the LRM this is actually supposed to crash the program.
164      fprintf(stderr, "EXITING — Attempted to close something that was not a valid file
             pointer\n");
165      exit(-1);
166    }
167    int fileNum = (int) file_handle->numericVal;
168
169    if (fileNum > open_num_files || open_files[fileNum] == NULL) {
170      // Per the LRM this is actually supposed to crash the program.
171      fprintf(stderr, "EXITING — Attempted to close something that was not a valid file
             pointer\n");
```

```
172      exit(−1);
173    }
174    fclose(open_files[fileNum]);
175    open_files[fileNum] = NULL; // Empty the container for the pointer.
176    return new_val(); // asssuming it was an open valid handle, close() is just supposed
            to return empty
177  }
178
179  value_p extend_read(value_p file_handle, value_p num_bytes){
180    if(!assertSingleNumber(file_handle) || !assertSingleNumber(num_bytes)) return
          new_val();
181    int max_bytes;
182    int fileNum = (int) file_handle−>numericVal;
183    if (fileNum > open_num_files || open_files[fileNum] == NULL)  return new_val();
184    FILE *f = open_files[fileNum];
185    max_bytes = (int) num_bytes−>numericVal;
186    if (max_bytes == 0) {
187      long cur_pos = ftell(f);
188      fseek(f, 0, SEEK_END);
189      long end_pos = ftell(f);
190      fseek(f, cur_pos, SEEK_SET);
191      max_bytes = end_pos − cur_pos;
192    }
193    char *buf = malloc(sizeof(char) * (max_bytes + 1));
194    int bytes_read = fread(buf, sizeof(char), max_bytes, f);
195    buf[bytes_read] = 0;
196    value_p result = new_string(buf);
197    free(buf);
198    return result;
199    //edge case: how to return the entire contents of the file if n == empty?
200  }
201
202  value_p extend_readline(value_p file_handle) {
203    int i=0, buf_size = 256;
204    char next_char;
205    if (!assertSingleNumber(file_handle)) return new_val();
206    int fileNum = (int) file_handle−>numericVal;
207    FILE *f = open_files[fileNum];
208    if (fileNum > open_num_files || open_files[fileNum] == NULL) {
209      return new_val();
210    }
211    char *buf = (char *) malloc (buf_size * sizeof(char));
212    while ((next_char = fgetc(f)) != '\n') {
213      buf[i++] = next_char;
214      if (i == buf_size − 2) {
215        buf_size *= 2;
216        char *new_buf = (char *) malloc (buf_size * sizeof(char));
217        memcpy(new_buf, buf, i);
218        free(buf);
219        buf = new_buf;
220      }
221    }
222    buf[i] = '\0';
223    value_p result = new_string(buf);
224    free(buf);
225    return result;
```

```
226  }
227
228  value_p extend_write(value_p file_handle, value_p buffer){
229    if(!assertSingleNumber(file_handle) || !assertSingleString(buffer)) return new_val()
           ;
230    int fileNum = (int) file_handle->numericVal;
231    if (fileNum > open_num_files || open_files[fileNum] == NULL) {
232      // Per the LRM this is actually supposed to crash the program.
233      fprintf(stderr, "EXITING - Attempted to write to something that was not a valid
             file pointer\n");
234      exit(-1);
235    }
236    fwrite(buffer->str->text, 1, buffer->str->length, open_files[fileNum]);
237    // TODO: make this return empty once compiler handles Id(s)
238    // RN: Use the return value to close the file
239    return new_number((double) fileNum);
240  }
241
242  value_p extend_current_hour() {
243    time_t ltime;
244    struct tm info;
245    ltime = time(&ltime);
246    localtime_r(&ltime, &info);
247    return new_number((double) info.tm_hour);
248  }
249
250  value_p extend_isNaN(value_p val) {
251    if (!assertSingleNumber(val)) return new_val();
252    double d = val->numericVal;
253    return isnan(d) ? new_number(1.0) : new_number(0.0);
254  }
255
256  value_p extend_isInfinite(value_p val) {
257    if (!assertSingleNumber(val)) return new_val();
258    double d = val->numericVal;
259    if (isinf(d)) {
260        return d < 0 ? new_number(-1.0) : new_number(1.0);
261    } else {
262      return new_number(0.0);
263    }
264  }
265
266  value_p extend_toASCII(value_p val) {
267    if (!assertSingleString(val)) return new_val();
268    value_p *val_arr = malloc(sizeof(value_p) * val->str->length);
269    int i;
270    for(i = 0; i < val->str->length; i++) {
271      value_p my_val = malloc(sizeof(struct value_t));
272      my_val->flags = FLAG_NUMBER;
273      my_val->numericVal = (double)val->str->text[i];
274      val_arr[i] = my_val;
275    }
276    value_p _new = new_subrange(1,val->str->length, val_arr);
277    return _new;
278  }
279
```

```
280  value_p extend_fromASCII(value_p val) {
281    value_p result = new_val();
282    if(val→flags == FLAG_NUMBER) {
283      char xxx = ((char)lrint(val→numericVal));
284      setString(result, &xxx, 1);
285    }
286    else if(val→flags == FLAG_SUBRANGE) {
287      int rows, cols, len;
288      rows = val→subrange→subrange_num_rows;
289      cols = val→subrange→subrange_num_cols;
290      if(rows > 1 && cols > 1) return result;
291      else len = rows == 1 ? cols : rows;
292      char *text = malloc(sizeof(char) * len);
293      for(rows = 0; rows < val→subrange→subrange_num_rows; rows++) {
294        for(cols = 0; cols < val→subrange→subrange_num_cols; cols++) {
295          value_p single = getValSR(val→subrange, rows, cols);
296          if(single→flags != FLAG_NUMBER) {
297            free(text);
298            return result;
299          }
300          text[rows + cols] = (char)lrint(single→numericVal);
301        }
302      }
303      setString(result, text, len);
304    }
305    return result;
306  }
```

7.11 runtime.c

```
 1  #include<stdio.h>
 2  #include<stdlib.h>
 3  #include<math.h>
 4  #include<sys/resource.h>
 5  #include<string.h>
 6  #include<stdbool.h>
 7  #include "runtime.h"
 8
 9  struct value_t zero_val = {FLAG_NUMBER, 0.0, NULL, NULL};
10  struct value_t one_val = {FLAG_NUMBER, 1.0, NULL, NULL};
11  struct rhs_index absolute_zero = {&zero_val, RHS_IDX_ABSOLUTE};
12  struct rhs_index absolute_one = {&one_val, RHS_IDX_ABSOLUTE};
13  struct rhs_slice zero_to_one = {&absolute_zero, &absolute_one};
14  struct rhs_slice corresponding_cell = {NULL, NULL};
15
16  void debug_print_subrange(subrange_p subrng);
17
18  void debug_print(value_p val, char *which_value) {
19    char *flag_meanings[4] = {"Empty", "Number", "String", "Subrange"};
20    fprintf(stderr, "————Everything you ever wanted to know about %s:————\n",
          which_value == NULL ? "some anonymous variable" : which_value);
21    fprintf(stderr, "Memory address: %p\n", val);
22    if (val == NULL) {
23      fprintf(stderr, "—————————Nice try asking me to dereference a null pointer\n
            ————————");
```

```
24        return;
25    }
26    fprintf(stderr, "Flags: %d (%s)\n", val->flags, flag_meanings[val->flags]);
27    fprintf(stderr, "NumericVal: %f\n", val->numericVal);
28    fprintf(stderr, "String contents: Probably safer not to check that pointer (%p)
          blindly\n", val->str);
29    if (val->flags == FLAG_STRING && val->str != NULL) {
30      fprintf(stderr, "It says it's a string and it's not a NULL pointer though, so here
            you go:\n");
31      fprintf(stderr, "String refcount: %d\n", val->str->refs);
32      fprintf(stderr, "String length: %ld\n", val->str->length);
33      fprintf(stderr, "String char* memory address: %p\n", val->str->text);
34      if (val->str->text == NULL) {
35        fprintf(stderr, "Not going to print the contents of NULL!\n");
36      } else {
37        fprintf(stderr, "String char* contents:\n%s\n", val->str->text);
38      }
39    }
40    fprintf(stderr, "Subrange contents: Probably safer not to check that pointer (%p)
          blindly either\n", val->subrange);
41    if (val->flags == FLAG_SUBRANGE && val->subrange != NULL) {
42      fprintf(stderr, "It says it's a subrange and it's not a NULL pointer though, so
            here you go:\n");
43      debug_print_subrange(val->subrange);
44    }
45    fprintf(stderr, "————That's all I've got to say about %s:————\n", which_value ==
          NULL ? "some anonymous variable" : which_value);
46 }
47
48 void debug_print_formula(struct ExtendFormula *fdef) {
49    fprintf(stderr, "————Everything you ever wanted to know about your favorite
          formula:————\n");
50    fprintf(stderr, "RowStart varnum: %d %d\n", fdef->rowStart_varnum, fdef->
          fromFirstRow);
51    fprintf(stderr, "RowEnd varnum: %d %d\n", fdef->rowEnd_varnum, fdef->toLastRow);
52    fprintf(stderr, "ColStart varnum: %d %d\n", fdef->colStart_varnum, fdef->
          fromFirstCol);
53    fprintf(stderr, "ColEnd varnum: %d %d\n", fdef->colEnd_varnum, fdef->toLastCol);
54 }
55
56 void debug_print_res_formula(struct ResolvedFormula *rdef) {
57    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);
58 }
59
60 void debug_print_vardefn(struct var_defn *pdef) {
61    fprintf(stderr, "————Everything you ever wanted to know about var defn %s:————\n
          ", pdef->name);
62    fprintf(stderr, "Row varnum: %d\n", pdef->rows_varnum);
63    fprintf(stderr, "Col varnum: %d\n", pdef->cols_varnum);
64    fprintf(stderr, "Num formulas: %d\n", pdef->numFormulas);
65    fprintf(stderr, "Formula defs: \n");
66    int i;
67    for (i=0; i < pdef->numFormulas; i++) {
68      debug_print_formula(pdef->formulas + i);
69    }
```

```
70      fprintf(stderr, "Is 1x1: %d\n", pdef->isOneByOne);
71    }
72
73    void debug_print_varinst(struct var_instance *inst) {
74      fprintf(stderr, "————Everything you ever wanted to know about var %s:————\n",
            inst->name);
75      fprintf(stderr, "Rows: %d\n", inst->rows);
76      fprintf(stderr, "Cols: %d\n", inst->cols);
77      fprintf(stderr, "Num formulas: %d\n", inst->numFormulas);
78      fprintf(stderr, "*****Formulas:*****\n");
79      int i;
80      for (i = 0; i < inst->numFormulas; i++) {
81        debug_print_res_formula(inst->formulas + i);
82      }
83      fprintf(stderr, "**** End of Formulas *** \n");
84      fprintf(stderr, "~~~~~~~~Cells:~~~~~~~~\n");
85      fprintf(stderr, "Status memory address: %p\n", inst->status);
86      for (i = 0; i < inst->rows * inst->cols; i++) {
87        printf("%s[%d,%d]: Status=%d\n", inst->name, i / inst->cols, i % inst->cols, inst
            ->status[i]);
88        if (inst->status[i] == CALCULATED) {
89          printf("%s[%d,%d] Value:\n", inst->name, i / inst->cols, i % inst->cols);
90          debug_print(inst->values[i], inst->name);
91        }
92      }
93      fprintf(stderr, "~~~ End of Cells: ~~~\n");
94    }
95
96    void debug_print_subrange(subrange_p subrng) {
97      fprintf(stderr, "————Everything you wanted to know about this subrange————\n");
98      fprintf(stderr, "Offset: [%d,%d]\n", subrng->base_var_offset_row, subrng->
            base_var_offset_col);
99      fprintf(stderr, "Dimensions: [%d,%d]\n", subrng->subrange_num_rows, subrng->
            subrange_num_cols);
100     fprintf(stderr, "Subrange of: \n");
101     debug_print_varinst(subrng->range);
102   }
103
104   void debug_print_index(struct rhs_index *idx) {
105     if (idx == NULL) {
106       fprintf(stderr, "I'd rather not try to print out the contents of a NULL index.\n")
              ;
107       exit(-1);
108     }
109     fprintf(stderr, "Index type: ");
110     switch(idx->rhs_index_type) {
111       case RHS_IDX_ABSOLUTE:
112         fprintf(stderr, "Absolute\n");
113         if (idx->val_of_expr == NULL) {
114           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");
115         } else {
116           debug_print(idx->val_of_expr, "an absolute index");
117         }
118         break;
119       case RHS_IDX_RELATIVE:
```

```
120        fprintf(stderr, "Relative\n");
121        if (idx->val_of_expr == NULL) {
122          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");
123        } else {
124          debug_print(idx->val_of_expr, "a relative index");
125        }
126        break;
127      case RHS_IDX_DIM_START:
128        fprintf(stderr, "DimensionStart\n");
129        if (idx->val_of_expr != NULL) {
130          fprintf(stderr, "This definitely isn't supposed to happen — the value pointer
                    isn't NULL. You should look into that.\n");
131          exit(-1);
132        }
133        break;
134      case RHS_IDX_DIM_END:
135        fprintf(stderr, "DimensionEnd\n");
136        if (idx->val_of_expr != NULL) {
137          fprintf(stderr, "This definitely isn't supposed to happen — the value pointer
                    isn't NULL. You should look into that.\n");
138          exit(-1);
139        }
140        break;
141    }
142 }
143
144 void debug_print_slice(struct rhs_slice *sl) {
145    if (sl == NULL) {
146      fprintf(stderr, "I'd rather not try to print out the contents of a NULL slice.\n")
                ;
147      exit(-1);
148    }
149    fprintf(stderr, "————Everything about this slice————\n");
150    fprintf(stderr, "Start and end index memory addresses: %p and %p\n", sl->
            slice_start_index, sl->slice_end_index);
151    if (sl->slice_start_index != NULL) {
152      fprintf(stderr, "Start index info:\n");
153      debug_print_index(sl->slice_start_index);
154      if (sl->slice_end_index != NULL) {
155        fprintf(stderr, "End index info:\n");
156        debug_print_index(sl->slice_end_index);
157      }
158    } else {
159      if (sl->slice_end_index != NULL) {
160        fprintf(stderr, "Start index is NULL but end index is not NULL. That should
                  never happen.\n");
161        fprintf(stderr, "Attempting to print contents anyway:\n");
162        fflush(stderr);
163        debug_print_index(sl->slice_end_index);
164      }
165    }
166 }
167
168 void debug_print_selection(struct rhs_selection *sel) {
169    if (sel == NULL) {
```

```
170        fprintf(stderr, "I'd rather not try to print out the contents of a NULL selection
              .\n");
171        exit(-1);
172      }
173      fprintf(stderr, "————————Everything about this selection————————\n");
174      fprintf(stderr, "Slice memory addresses: %p and %p\n", sel->slice1, sel->slice2);
175      if (sel->slice1 != NULL) {
176        fprintf(stderr, "Slice 1 info:\n");
177        debug_print_slice(sel->slice1);
178        if (sel->slice2 != NULL) {
179          fprintf(stderr, "Slice 2 info:\n");
180          debug_print_slice(sel->slice2);
181        }
182      } else {
183        if (sel->slice2 != NULL) {
184          fprintf(stderr, "Slice 1 is NULL but slice 2 is not NULL. That should never
                happen.\n");
185          fprintf(stderr, "Attempting to print contents anyway:\n");
186          fflush(stderr);
187          debug_print_slice(sel->slice2);
188        }
189      }
190      fprintf(stderr, "————————That's all I've got about that selection————————\n\n");
191    }
192
193    int rg_eq(value_p val1, value_p val2) {
194      int res = 1;
195      if(val1->flags != val2->flags) res = 0;
196      else if(val1->flags == FLAG_EMPTY) ;
197      else if(val1->flags == FLAG_NUMBER && val1->numericVal != val2->numericVal) res = 0;
198      else if(val1->flags == FLAG_STRING && strcmp(val1->str->text, val2->str->text)) res
              = 0;
199      else if(val1->flags == FLAG_SUBRANGE) {
200        subrange_p sr1 = val1->subrange;
201        subrange_p sr2 = val2->subrange;
202        if(sr1->subrange_num_cols != sr2->subrange_num_cols || sr1->subrange_num_rows !=
              sr2->subrange_num_rows) {
203          return 0;
204        } else {
205          int i, j;
206          value_p v1, v2;
207          for(i = 0; i < sr1->subrange_num_rows; i++) {
208            for(j = 0; j < sr1->subrange_num_cols; j++) {
209              v1 = getValSR(sr1, i, j);
210              v2 = getValSR(sr2, i, j);
211              if(rg_eq(v1, v2) == 0) {
212                return 0;
213              }
214            }
215          }
216        }
217      }
218      return res;
219    }
220
221    void incStack() {
```

```
222    const rlim_t kStackSize = 64L * 1024L * 1024L;
223    struct rlimit rl;
224    int result;
225
226    result = getrlimit(RLIMIT_STACK, &rl);
227    rl.rlim_cur = rl.rlim_max;
228    result = setrlimit(RLIMIT_STACK, &rl);
229 }
230
231 double setNumeric(value_p result, double val) {
232    result->flags = FLAG_NUMBER;
233    return (result->numericVal = val);
234 }
235
236 char* setString(value_p result, char *str, int length) {
237    result->flags = FLAG_STRING;
238    result->str = malloc(sizeof(struct string_t));
239    result->str->length = length;
240    return (result->str->text = str);
241 }
242
243 double setFlag(value_p result, double flag_num) {
244    return (result->flags = flag_num);
245 }
246
247 int assertSingle(value_p value) {
248    /* TODO: dereference 1 by 1 subrange */
249    return !(value->flags == FLAG_SUBRANGE);
250 }
251
252 int assertSingleNumber(value_p p) {
253    if (!assertSingle(p)) {
254       return 0;
255    }
256    return (p->flags == FLAG_NUMBER);
257 }
258
259 int assertText(value_p my_val) {
260    return (my_val->flags == FLAG_STRING);
261 }
262
263 int assertSingleString(value_p p) {
264    if (!assertSingle(p)) {
265       return 0;
266    }
267    return (p->flags == FLAG_STRING);
268 }
269
270 int assertEmpty(value_p p) {
271    if (!assertSingle(p)) {
272       return 0;
273    }
274    return (p->flags == FLAG_EMPTY);
275 }
276
277 value_p new_val() {
```

```
278     value_p empty_val = malloc(sizeof(struct value_t));
279     setFlag(empty_val, FLAG_EMPTY);
280     return empty_val;
281   }
282
283   value_p new_number(double val) {
284     value_p new_v = malloc(sizeof(struct value_t));
285     setFlag(new_v, FLAG_NUMBER);
286     setNumeric(new_v, val);
287     return new_v;
288   }
289
290   value_p new_string(char *s) {
291     if (s == NULL) return new_val();
292     value_p new_v = malloc(sizeof(struct value_t));
293     setFlag(new_v, FLAG_STRING);
294     string_p new_str = malloc(sizeof(struct string_t));
295     long len = strlen(s);
296     new_str->text = malloc(len+1);
297     strcpy(new_str->text, s);
298     new_str->length = len;
299     new_str->refs = 1;
300     new_v->str = new_str;
301     return new_v;
302   }
303
304   struct ExtendScope *global_scope;
305
306   void null_init(struct ExtendScope *scope_ptr) {
307     int i;
308     for(i = 0; i < scope_ptr->numVars; i++)
309       scope_ptr->vars[i] = NULL;
310   }
311
312   int getIntFromOneByOne(struct ExtendScope *scope_ptr, int varnum) {
313     if (!scope_ptr->defns[varnum].isOneByOne) {
314       fprintf(stderr, "The variable you claimed (%s) was one by one is not defined that
               way.\n", scope_ptr->defns[varnum].name);
315     }
316     struct var_instance *inst = get_variable(scope_ptr, varnum);
317     if (inst->rows != 1 || inst->cols != 1) {
318       fprintf(stderr, "The variable you claimed (%s) was one by one is actually %d by %d
               .\n", inst->name, inst->rows, inst->cols);
319       debug_print_varinst(inst);
320       exit(-1);
321     }
322     value_p val = getVal(inst, 0, 0);
323     if (!assertSingleNumber(val)) {
324       fprintf(stderr, "The variable you claimed (%s) was a number isn't.\n", inst->name)
               ;
325       debug_print(val, inst->name);
326       exit(-1);
327     }
328     return (int) lrint(val->numericVal);
329   }
330
```

```
331  struct var_instance *instantiate_variable(struct ExtendScope *scope_ptr, struct
        var_defn def) {
332    struct var_instance *inst = malloc(sizeof(struct var_instance));
333    if(def.isOneByOne) {
334      inst->rows = 1;
335      inst->cols = 1;
336    } else {
337      inst->rows = getIntFromOneByOne(scope_ptr, def.rows_varnum);
338      inst->cols = getIntFromOneByOne(scope_ptr, def.cols_varnum);
339    }
340    // TODO: do the same thing for each FormulaFP to turn an ExtendFormula into a
        ResolvedFormula
341    inst->numFormulas = def.numFormulas;
342    inst->closure = scope_ptr;
343    inst->name = def.name;
344    int size = inst->rows * inst->cols;
345    inst->values = malloc(sizeof(value_p) * size);
346    memset(inst->values, 0, sizeof(value_p) * size);
347    inst->status = malloc(sizeof(char) * size);
348    memset(inst->status, 0, sizeof(char) * size);
349    inst->formulas = malloc(sizeof(struct ResolvedFormula) * inst->numFormulas);
350    //debug_print_vardefn(&def);
351    //debug_print_varinst(inst);
352    int i, j;
353    for(i = 0; i < inst->numFormulas; i++) {
354
355      // Set the formula function pointer to the pointer from the definition
356      inst->formulas[i].formula = def.formulas[i].formula;
357
358      if (def.isOneByOne) {
359        inst->formulas[i].rowStart = 0;
360        inst->formulas[i].rowEnd = 1;
361        inst->formulas[i].colStart = 0;
362        inst->formulas[i].colEnd = 1;
363      } else {
364        if(def.formulas[i].fromFirstRow) {
365          inst->formulas[i].rowStart = 0;
366        } else {
367          inst->formulas[i].rowStart = getIntFromOneByOne(scope_ptr, def.formulas[i].
              rowStart_varnum);
368          if (inst->formulas[i].rowStart < 0) {
369            inst->formulas[i].rowStart += inst->rows;
370          }
371          if (inst->formulas[i].rowStart < 0 || inst->formulas[i].rowStart >= inst->rows
              ) {
372            //Doesn't matter, but will never get called
373          }
374        }
375        if (def.formulas[i].isSingleRow) {
376          inst->formulas[i].rowEnd = inst->formulas[i].rowStart + 1;
377        } else if (def.formulas[i].toLastRow) {
378          inst->formulas[i].rowEnd = inst->rows;
379        } else {
380          inst->formulas[i].rowEnd = getIntFromOneByOne(scope_ptr, def.formulas[i].
              rowEnd_varnum);
381          if (inst->formulas[i].rowEnd < 0) {
```

```
382          inst->formulas[i].rowEnd += inst->rows;
383        }
384      }
385      if(def.formulas[i].fromFirstCol) {
386        inst->formulas[i].colStart = 0;
387      } else {
388        inst->formulas[i].colStart = getIntFromOneByOne(scope_ptr, def.formulas[i].
               colStart_varnum);
389        if (inst->formulas[i].colStart < 0) {
390          inst->formulas[i].colStart += inst->cols;
391        }
392        if (inst->formulas[i].colStart < 0 || inst->formulas[i].colStart >= inst->cols
             ) {
393          //Doesn't matter, but will never get called
394        }
395      }
396      if (def.formulas[i].isSingleCol) {
397        inst->formulas[i].colEnd = inst->formulas[i].colStart + 1;
398      } else if (def.formulas[i].toLastCol) {
399        inst->formulas[i].colEnd = inst->cols;
400      } else {
401        inst->formulas[i].colEnd = getIntFromOneByOne(scope_ptr, def.formulas[i].
               colEnd_varnum);
402        if (inst->formulas[i].colEnd < 0) {
403          inst->formulas[i].colEnd += inst->cols;
404        }
405      }
406    }
407  }
408
409  for (i = 1; i < inst->numFormulas; i++) {
410    for (j = 0; j < i; j++) {
411      int intersectRowStart = (inst->formulas[i].rowStart > inst->formulas[j].rowStart
             ) ? inst->formulas[i].rowStart : inst->formulas[j].rowStart;
412      int intersectColStart = (inst->formulas[i].colStart > inst->formulas[j].colStart
             ) ? inst->formulas[i].colStart : inst->formulas[j].colStart;
413      int intersectRowEnd = (inst->formulas[i].rowEnd < inst->formulas[j].rowEnd) ?
             inst->formulas[i].rowEnd : inst->formulas[j].rowEnd;
414      int intersectColEnd = (inst->formulas[i].colEnd < inst->formulas[j].colEnd) ?
             inst->formulas[i].colEnd : inst->formulas[j].colEnd;
415      if (intersectRowEnd > intersectRowStart && intersectColEnd > intersectColStart)
             {
416        fprintf(stderr, "Runtime error: Multiple formulas were assigned to %s[%d:%d,%d
               :%d].\n", inst->name,
417                        intersectRowStart, intersectRowEnd, intersectColStart,
                            intersectColEnd);
418        exit(-1);
419      }
420    }
421  }
422
423  scope_ptr->refcount++;
424  return inst;
425 }
426
427 struct var_instance *get_variable(struct ExtendScope *scope_ptr, int varnum) {
```

```
428    if (varnum >= scope_ptr->numVars) {
429      fprintf(stderr, "Runtime error: Asked for nonexistant variable number\n");
430      exit(-1);
431    }
432    if (scope_ptr->vars[varnum] == NULL) {
433      scope_ptr->vars[varnum] = instantiate_variable(scope_ptr, scope_ptr->defns[varnum
           ]);
434    }
435    return scope_ptr->vars[varnum];
436  }
437
438  char assertInBounds(struct var_instance *defn, int r, int c) {
439    return (
440      r >= 0 && r < defn->rows &&
441      c >= 0 && c < defn->cols
442    );
443  }
444
445  value_p calcVal(struct var_instance *inst, int r, int c) {
446    int i;
447    for (i = 0; i < inst->numFormulas; i++) {
448      if (
449        r >= inst->formulas[i].rowStart && r < inst->formulas[i].rowEnd &&
450        c >= inst->formulas[i].colStart && c < inst->formulas[i].colEnd
451      ) {
452        return (inst->formulas[i].formula)(inst->closure, r, c);
453      }
454    }
455    return new_val();
456  }
457
458  value_p clone_value(value_p old_value) {
459    value_p new_value = (value_p) malloc(sizeof(struct value_t));
460    new_value->flags = old_value->flags;
461    switch (new_value->flags) {
462      case FLAG_EMPTY:
463        break;
464      case FLAG_NUMBER:
465        new_value->numericVal = old_value->numericVal;
466        break;
467      case FLAG_STRING:
468        new_value->str = old_value->str;
469        new_value->str->refs++;
470        break;
471      case FLAG_SUBRANGE:
472        new_value->subrange = (subrange_p) malloc(sizeof(struct subrange_t));
473        memcpy(new_value->subrange, old_value->subrange, sizeof(struct subrange_t));
474        if (new_value->subrange->range->closure != NULL) {
475          new_value->subrange->range->closure->refcount++; /* Not sure about this one */
476        }
477        break;
478      default:
479        fprintf(stderr, "clone_value(%p): Illegal value of flags: %c\n", old_value,
             new_value->flags);
480        exit(-1);
481        break;
```

```c
482    }
483    return new_value;
484  }
485
486  void delete_string_p(string_p old_string) {
487    old_string->refs--;
488    if (old_string->refs == 0) {
489      /* free(old_string); */
490    }
491  }
492
493  void delete_subrange_p(subrange_p old_subrange) {
494    if (old_subrange->range->closure != NULL) {
495      old_subrange->range->closure->refcount--;
496    }
497    free(old_subrange);
498  }
499
500  void delete_value(value_p old_value) {
501    switch (old_value->flags) {
502      case FLAG_EMPTY:
503        break;
504      case FLAG_NUMBER:
505        break;
506      case FLAG_STRING:
507        delete_string_p(old_value->str); /* doesn't do anything besides decrement the
             ref count now */
508        break;
509      case FLAG_SUBRANGE:
510        delete_subrange_p(old_value->subrange);
511        break;
512      default:
513        fprintf(stderr, "delete_value(%p): Illegal value of flags: %c\n", old_value,
             old_value->flags);
514        exit(-1);
515        break;
516    }
517  }
518
519  value_p deref_subrange_p(subrange_p subrng) {
520    if (subrng == NULL) {
521      fprintf(stderr, "Exiting - asked to dereference a NULL pointer.\n");
522      exit(-1);
523    }
524    if (subrng->subrange_num_rows == 1 && subrng->subrange_num_cols == 1) {
525      return getVal(subrng->range, subrng->base_var_offset_row, subrng->
             base_var_offset_col);
526    } else {
527      value_p new_value = (value_p) malloc (sizeof(struct value_t));
528      new_value->flags = FLAG_SUBRANGE;
529      new_value->numericVal = 0.0;
530      new_value->str = NULL;
531      new_value->subrange = (subrange_p) malloc (sizeof(struct subrange_t));
532      memcpy(new_value->subrange, subrng, sizeof(struct subrange_t));
533      if (new_value->subrange->range->closure != NULL) {
534        new_value->subrange->range->closure->refcount++;
```

```
535      }
536      return new_value;
537    }
538  }
539
540  value_p new_subrange(int num_rows, int num_cols, value_p *vals) {
541    /* This function does not check its arguments; if you supply fewer
542     * than num_rows * num_cols elements in vals, it will crash.
543     * Only use this function if you know what you're doing. */
544    struct subrange_t sr;
545    sr.range = (struct var_instance *) malloc (sizeof(struct var_instance));
546    sr.base_var_offset_row = 0;
547    sr.base_var_offset_col = 0;
548    sr.subrange_num_rows = num_rows;
549    sr.subrange_num_cols = num_cols;
550    sr.range->rows = num_rows;
551    sr.range->cols = num_cols;
552    sr.range->numFormulas = 0;
553    sr.range->formulas = NULL;
554    sr.range->closure = NULL;
555    sr.range->values = (value_p *) malloc(num_rows * num_cols * sizeof(value_p));
556    sr.range->status = (char *) malloc (num_rows * num_cols * sizeof(char));
557    sr.range->name = NULL;
558    int i;
559    for (i = 0; i < num_rows * num_cols; i++) {
560      sr.range->values[i] = clone_value(vals[i]);
561      sr.range->status[i] = CALCULATED;
562    }
563    return deref_subrange_p(&sr);
564  }
565
566  value_p box_command_line_args(int argc, char **argv) {
567    value_p *vals = (value_p *) malloc (argc * sizeof(value_p));
568    int i;
569    for (i = 0; i < argc; i++) {
570      vals[i] = new_string(argv[i]);
571    }
572    value_p ret = new_subrange(1, argc, vals);
573    for (i = 0; i < argc; i++) {
574      free(vals[i]);
575    }
576    free(vals);
577    return ret;
578  }
579
580  char resolve_rhs_index(struct rhs_index *index, int dimension_len, int
         dimension_cell_num, int *result_ptr) {
581    if (index == NULL) {
582      fprintf(stderr, "Exiting — asked to dereference a NULL index\n");
583      exit(-1);
584    }
585    int i;
586    switch(index->rhs_index_type) {
587      case RHS_IDX_ABSOLUTE:
588        if (!assertSingleNumber(index->val_of_expr)) return false;
589        i = (int) lrint(index->val_of_expr->numericVal);
```

```
590        if (i >= 0) {
591          *result_ptr = i;
592        } else {
593          *result_ptr = i + dimension_len;
594        }
595        return true;
596        break;
597      case RHS_IDX_RELATIVE:
598        if (!assertSingleNumber(index->val_of_expr)) return false;
599        *result_ptr = dimension_cell_num + (int) lrint(index->val_of_expr->numericVal);
600        return true;
601        break;
602      case RHS_IDX_DIM_START:
603        *result_ptr = 0;
604        return true;
605        break;
606      case RHS_IDX_DIM_END:
607        *result_ptr = dimension_len;
608        return true;
609        break;
610      default:
611        fprintf(stderr, "Exiting — illegal index type\n");
612        exit(-1);
613        break;
614    }
615  }
616
617  char resolve_rhs_slice(struct rhs_slice *slice, int dimension_len, int
        dimension_cell_num, int *start_ptr, int *end_ptr) {
618    char start_success, end_success;
619    if (slice == NULL) {
620      fprintf(stderr, "Exiting — asked to dereference a NULL slice\n");
621      exit(-1);
622    }
623    if (slice->slice_start_index == NULL) {
624      if (slice->slice_end_index != NULL) {
625        fprintf(stderr, "Exiting — illegal slice\n");
626        exit(-1);
627      }
628      if (dimension_len == 1) {
629        *start_ptr = 0;
630        *end_ptr = 1;
631        return true;
632      } else {
633        *start_ptr = dimension_cell_num;
634        *end_ptr = dimension_cell_num + 1;
635        return true;
636      }
637    } else {
638      start_success = resolve_rhs_index(slice->slice_start_index, dimension_len,
          dimension_cell_num, start_ptr);
639      if (!start_success) return false;
640      if (slice->slice_end_index == NULL) {
641        *end_ptr = *start_ptr + 1;
642        return true;
643      } else {
```

```
644        end_success = resolve_rhs_index(slice->slice_end_index, dimension_len,
                dimension_cell_num, end_ptr);
645        return end_success;
646      }
647    }
648  }
649
650  value_p extract_selection(value_p expr, struct rhs_selection *sel, int r, int c) {
651    int expr_rows, expr_cols;
652    struct subrange_t subrange;
653    struct rhs_slice *row_slice_p, *col_slice_p;
654    int row_start, row_end, col_start, col_end;
655    char row_slice_success, col_slice_success;
656
657    if (expr == NULL || sel == NULL) {
658      fprintf(stderr, "Exiting - asked to extract a selection using a NULL pointer.\n");
659      exit(-1);
660    }
661    switch(expr->flags) {
662      case FLAG_EMPTY:
663        return new_val();
664        break;
665      case FLAG_NUMBER: case FLAG_STRING:
666        expr_rows = 1;
667        expr_cols = 1;
668        break;
669      case FLAG_SUBRANGE:
670        expr_rows = expr->subrange->subrange_num_rows;
671        expr_cols = expr->subrange->subrange_num_cols;
672        break;
673      default:
674        fprintf(stderr, "Exiting - invalid value type\n");
675        exit(-1);
676        break;
677    }
678    if (sel->slice1 == NULL) {
679      if (sel->slice2 != NULL) {
680        fprintf(stderr, "Exiting - illegal selection\n");
681        exit(-1);
682      }
683      row_slice_p = &corresponding_cell;
684      col_slice_p = &corresponding_cell;
685    } else {
686      if (sel->slice2 == NULL) {
687        if (expr_rows == 1) {
688          row_slice_p = &zero_to_one;
689          col_slice_p = sel->slice1;
690        } else if (expr_cols == 1) {
691          row_slice_p = sel->slice1;
692          col_slice_p = &zero_to_one;
693        } else {
694          return new_val();
695  /*      Alternately:
696          fprintf(stderr, "Runtime error: Only given one slice for a value with multiple
                  rows and multiple columns\n");
697          debug_print(expr);
```

107

```
698         exit(-1); */
699       }
700     } else {
701       row_slice_p = sel->slice1;
702       col_slice_p = sel->slice2;
703     }
704   }
705   row_slice_success = resolve_rhs_slice(row_slice_p, expr_rows, r, &row_start, &
          row_end);
706   col_slice_success = resolve_rhs_slice(col_slice_p, expr_cols, c, &col_start, &
          col_end);
707   if (!row_slice_success || !col_slice_success) return new_val();
708   if (row_start < 0) row_start = 0;
709   if (col_start < 0) col_start = 0;
710   if (row_end > expr_rows) row_end = expr_rows;
711   if (col_end > expr_cols)  col_end = expr_cols;
712   if (row_end <= row_start || col_end <= col_start) return new_val();
713   if (expr->flags == FLAG_NUMBER || expr->flags == FLAG_STRING) {
714     /* You would have thought we could figure this out a lot further up
715      * in the code, but had to be sure that (row_start, row_end, col_start, col_end)
716      * actually ended up as (0, 1, 0, 1) */
717     return clone_value(expr);
718   } else {
719     subrange.range = expr->subrange->range;
720     subrange.base_var_offset_row = expr->subrange->base_var_offset_row + row_start;
721     subrange.base_var_offset_col = expr->subrange->base_var_offset_col + col_start;
722     subrange.subrange_num_rows = row_end - row_start;
723     subrange.subrange_num_cols = col_end - col_start;
724     return deref_subrange_p(&subrange);
725   }
726 }
727
728 value_p getValSR(struct subrange_t *sr, int r, int c) {
729   if(sr->subrange_num_rows <= r || sr->subrange_num_cols <= c || r < 0 || c < 0)
730     return new_val();
731   return getVal(sr->range, r + sr->base_var_offset_row, c + sr->base_var_offset_col);
732 }
733
734 void verify_assert(value_p val, char *fname) {
735   if ((!assertSingleNumber(val)) || val->numericVal != 1.0) {
736     fprintf(stderr, "EXITING - The function %s was called with arguments of the wrong
          dimensions.\n", fname);
737     exit(-1);
738   }
739 }
740
741 value_p getVal(struct var_instance *inst, int r, int c) {
742   /* If we're going to return new_val() then we have to
743    * do clone_value(). Otherwise the receiver won't know
744    * whether or not they can free the value_p they get back.
745    * I think this should return, dangerously, return NULL if it's
746    * invalid, and the callers will have to be careful to check the value.
747    * The alternative is to always clone_value - safer, but much slower
748    * and makes our memory issues even bigger.
749    * Right now there are only a few places that call this. */
750
```

```
751    if(!assertInBounds(inst, r, c)) return NULL;
752    int cell_number = r * inst->cols + c;
753    char cell_status = inst->status[cell_number];
754    switch(cell_status) {
755      case NEVER_EXAMINED:
756        inst->status[cell_number] = IN_PROGRESS;
757        inst->values[cell_number] = calcVal(inst, r, c);
758        inst->status[cell_number] = CALCULATED;
759        break;
760      case IN_PROGRESS:
761        fprintf(stderr, "EXITING - Circular reference in %s[%d,%d]\n", inst->name, r, c)
               ;
762        exit(-1);
763        break;
764      case CALCULATED:
765        if (inst->values[cell_number] == NULL) {
766          fprintf(stderr, "Supposedly, %s[%d,%d] was already calculated, but there is a
                 null pointer there.\n", inst->name, r, c);
767          fprintf(stderr, "Attempting to print contents of the variable instance where
                 this occurred:\n");
768          fflush(stderr);
769          debug_print_varinst(inst);
770          exit(-1);
771        }
772        break;
773      default:
774        fprintf(stderr, "Unrecognized cell status %d (row %d, col %d)!\n", cell_status,
               r, c);
775        fprintf(stderr, "Attempting to print contents of the variable instance where
                 this occurred:\n");
776        fflush(stderr);
777        debug_print_varinst(inst);
778        exit(-1);
779        break;
780    }
781    return inst->values[cell_number];
782 }
```

7.12 stdlib.xtnd

```
1  extern "stdlib.o" {
2    current_hour();
3    print(whatever, text);
4    printv(whatever, text);
5    printd(whatever, text);
6    to_string(val);
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    isInfinite(val);
25    get_stdin();
26    get_stdout();
27    get_stderr();
28    open(filename, mode);
29    close(file_handle);
30    read(file_handle, num_bytes);
31    readline(file_handle);
32    write(file_handle, buffer);
33    toASCII(val);
34    fromASCII(val);
35  }
36
37  global STDIN := get_stdin();
38  global STDOUT := get_stdout();
39  global STDERR := get_stderr();
40
41  print_endline(val) {
42    return write(STDOUT, to_string(val) + "\n");
43  }
```

8. Tests and Output

helloworld.xtnd

```
1  main(args) {
2    foo := printv(1,"Hello World\n") -> 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(1,to_string(bar)) -> print(1, "\n") -> 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(1,to_string( foo[1,0])+"\n") -> 0;
4  }
```

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

```
1  string
```

test-access-hashtag-multi-dim.xtnd

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

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

```
1  string
```

test-acos.xtnd

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

test-acos.xtnd - Expected Output

```
1  1.570796
```

test-addition.xtnd

```
1  main(args){
2    return print(1,to_string( 5 + 7)+"\n") -> 0;
3  }
```

test-addition.xtnd - Expected Output

```
1  12
```

test-addition-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty + 5)+"\n") -> 0;
3  }
```

test-addition-empty.xtnd - Expected Output

```
1  empty
```

test-asin.xtnd

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

test-asin.xtnd - Expected Output

```
1  0.523599
```

test-atan.xtnd

```
1  main([1,n] args) {
2    return printd(1, 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(1,to_string( 0)+"\n") -> 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(1,to_string( 23 & 12)+"\n") -> 0;
3  }
```

test-bitwise-and.xtnd - Expected Output

```
1  4
```

test-bitwise-and-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty & 4)+"\n") -> 0;
3  }
```

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

```
1  empty
```

test-bitwise-left.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 14 << 2)+"\n") -> 0;
3  }
```

test-bitwise-left.xtnd - Expected Output

```
1  56
```

test-bitwise-left-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty >> 1)+"\n") -> 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(1,to_string( ~88)+"\n") -> 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(1,to_string( ~empty)+"\n") -> 0;
4  }
```

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

```
1  empty
```

test-bitwise-or.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 14 | 12)+"\n") -> 0;
3  }
```

test-bitwise-or.xtnd - Expected Output

```
1  14
```

test-bitwise-or-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty | 2)+"\n") -> 0;
3  }
```

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

```
1  empty
```

test-bitwise-right.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 12 >> 2)+"\n") —> 0;
3  }
```

test-bitwise-right.xtnd - Expected Output

```
1  3
```

test-bitwise-right-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty >> 2)+"\n") —> 0;
3  }
```

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

```
1  empty
```

test-bitwise-xor.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 14 ^ 12)+"\n") —> 0;
3  }
```

test-bitwise-xor.xtnd - Expected Output

```
1  2
```

test-bitwise-xor-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty ^ 2)+"\n") —> 0;
3  }
```

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

```
1  empty
```

test-boolean-equals.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 5 == 6)+"\n") —> 0;
3  }
```

test-boolean-equals.xtnd - Expected Output

```
1  0
```

test-boolean-equals-both-empty.xtnd

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

```
56  }
```

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

```
 1  True cases for ==
 2  1.000000
 3  1.000000
 4  1.000000
 5  1.000000
 6  1.000000
 7  1.000000
 8  1.000000
 9  1.000000
10  1.000000
11  1.000000
12  1.000000
13
14  False cases for ==
15  0.000000
16  0.000000
17  0.000000
18  0.000000
19  0.000000
20  0.000000
21  0.000000
22  0.000000
23  0.000000
24  0.000000
25  0.000000
26
27  True cases for !=
28  1.000000
29  1.000000
30  1.000000
31  1.000000
32  1.000000
33  1.000000
34  1.000000
35  1.000000
36  1.000000
37  1.000000
38  1.000000
39
40  False cases for !=
41  0.000000
42  0.000000
43  0.000000
44  0.000000
45  0.000000
46  0.000000
47  0.000000
48  0.000000
49  0.000000
50  0.000000
51  0.000000
```

test-boolean-equals-one-empty.xtnd

117

```
1  main([1,1] args){
2    return print(1,to_string( empty == 5)+"\n") -> 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(1,to_string( 6 != 7)+"\n") -> 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(1,to_string( empty != empty)+"\n") -> 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(1,to_string( empty != 5)+"\n") -> 0;
3  }
```

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

```
1  1
```

test-calling-func-from-import.xtnd

```
1  import "../../samples/gcd_func.xtnd";
2
3  main([1,n] args){
4    return print(1,to_string( gcd(70, 55))+"\n") -> 0;
5  }
```

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

```
1  5
```

test-ceil.xtnd

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

test-ceil.xtnd - Expected Output

```
1  11.000000
```

test-cos.xtnd

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

test-cos.xtnd - Expected Output

```
1  0.525322
```

test-cosh.xtnd

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

test-cosh.xtnd - Expected Output

```
1  17467135528742547456.000000
```

test-division.xtnd

```
1  main([1,1] args){
2    /* Should evaluate to 4 */
3    return print(1,to_string( 20 / 5)+"\n") -> 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(1,to_string( empty / 5)+"\n") -> 0;
4  }
```

test-division-empty.xtnd - Expected Output

```
1  empty
```

test-exp.xtnd

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

test-exp.xtnd - Expected Output

```
1  7.389056
```

test-fabs.xtnd

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

test-fabs.xtnd - Expected Output

```
1  45.000000
```

119

test-file-close.xtnd

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

test-file-close.xtnd - Expected Output

```
1  Made it this far
```

test-file-read.xtnd

```
1  main(args){
2    return print(1, 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      print(1, 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(1,"Made it this far\n") ->
7      0;
8  }
```

test-file-write.xtnd - Expected Output

```
1  Made it this far
```

test-floor.xtnd

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

test-floor.xtnd - Expected Output

```
1  10.000000
```

test-func-params.xtnd

```
1  main([1,n] args) {
2    return print(1,to_string( foo("string"))+"\n") -> 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(1,to_string( foo("string"))+"\n") -> 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 := printv(1,"Hello Globals!\n") -> 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 := printv(1,"Hello Globals!\n") -> 0;
7
8  main(args) {
9    foo := printv(1,"Hello Locals!\n") -> 0;
10   return foo;
11 }
```

test-global-masking.xtnd - Expected Output

```
1  Hello Locals!
```

121

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(1,to_string( 6 > 5)+"\n") --> 0;
3  }
```

test-greater-than.xtnd - Expected Output

```
1  1
```

test-greater-than-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty > 5)+"\n") --> 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(1,to_string( 7 >= 7)+"\n") --> 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(1,to_string( empty >= 7)+"\n") --> 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(1,to_string( 6 < 7)+"\n") --> 0;
3  }
```

test-less-than.xtnd - Expected Output

```
1  1
```

test-less-than-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty > 5)+"\n") -> 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(1,to_string( 7 <= 5)+"\n") -> 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(1,to_string( empty <= 8)+"\n") -> 0;
3  }
```

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

```
1  empty
```

test-log.xtnd

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

test-log.xtnd - Expected Output

```
1  2.302585
```

test-log10.xtnd

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

test-log10.xtnd - Expected Output

```
1  2.000000
```

test-logical-and.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 1 && 6)+"\n") -> 0;
3  }
```

test-logical-and.xtnd - Expected Output

```
1  1
```

test-logical-and-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty && 1)+"\n") -> 0;
3  }
```

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

```
1  empty
```

test-logical-not.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( !5)+"\n") -> 0;
3  }
```

test-logical-not.xtnd - Expected Output

```
1  0
```

test-logical-not-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( !empty)+"\n") -> 0;
3  }
```

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

```
1  empty
```

test-logical-or.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( 5 || 6)+"\n") -> 0;
3  }
```

test-logical-or.xtnd - Expected Output

```
1  1
```

test-logical-or-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty || 4)+"\n") -> 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(1,to_string( 5 % 4)+"\n") -> 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(1,to_string( empty % 5)+"\n") -> 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(1,to_string( 7 * 5)+"\n") -> 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(1,to_string( empty * 5)+"\n") -> 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
```

127

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 printf(1,toString(x)+"\n") -> 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(1,to_string( 6**3)+"\n") -> 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(1,to_string( empty**5)+"\n") -> 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(1,to_string( foo)+"\n") -> 0;
4  }
```

test-print-empty.xtnd - Expected Output

```
1  empty
```

test-print-nums.xtnd

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

test-print-nums.xtnd - Expected Output

```
1  1
```

test-print-str.xtnd

```
1  main([1,n] args) {
2    foo := "string";
3    return print(1,to_string( foo)+"\n") -> 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(1,to_string( bar)+"\n") -> 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(1,"FAIL\n") -> print(1,"PASS\n") -> 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(1,"PASS1\n") -> print(1,"PASS2\n") -> 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(1,"PASS1\n") -> print(1,"PASS2\n") -> 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(1,"FAIL\n") -> print(1,"PASS\n") -> 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 " + to_string(m) + " rows and " +
         to_string(n) + " columns.";
3 }
4
5 main([1,1] args) {
6   [42,17] x;
7   return print(1,foo(x)+"\n") -> 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 printd(1, 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 printd(1, 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 printd(1, internal_sin(1,2,theta)) -> 0;
9  }
```

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

```
1  0.850904
```

test-single-import.xtnd

```
1  import "../../samples/gcd_func.xtnd";
2
3  main([1,n] args) {
4    return print(1, to_string(gcd(70, 55)) + "\n") -> 0;
5  }
```

test-single-import.xtnd - Expected Output

```
1  5
```

test-sinh.xtnd

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

test-sinh.xtnd - Expected Output

```
1  17467135528742547456.000000
```

test-sqrt.xtnd

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

test-sqrt.xtnd - Expected Output

```
1  3.000000
```

test-string-concatenation.xtnd

```
1  main(args) {
2    foo :=
3      printv(1,"Hello " + "World\n") ->
4      printv(1,"Hello " + "World" + "\n") ->
5      printv(1,("Hello " + "World") + ("" + "\n")) ->
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(1,to_string( 7 - 5)+"\n") -> 0;
3  }
```

test-subtraction.xtnd - Expected Output

```
1  2
```

test-subtraction-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty - 2)+"\n") -> 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(1,to_string(x)+"\n") -> 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 printd(1,x) -> 0;
8  }
```

test-switch-v10.xtnd - Expected Output

```
1  200.000000
```

test-switch-v11.xtnd

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

test-switch-v11.xtnd - Expected Output

```
1  99.000000
```

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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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(1,to_string(x)+"\n") -> 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 printd(1,x) -> 0;
7  }
```

test-switch-v9.xtnd - Expected Output

```
1  100.000000
```

test-tan.xtnd

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

test-tan.xtnd - Expected Output

```
1  1.619775
```

test-tanh.xtnd

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

test-tanh.xtnd - Expected Output

```
1  1.000000
```

test-ternary-conditional.xtnd

```
1  main([1,1] args){
2    return print(1,to_string(5 ? 2 : 3) + "\n") -> 0;
3  }
```

test-ternary-conditional.xtnd - Expected Output

```
1  2
```

test-ternary-conditional-empty.xtnd

```
1  main([1,1] args){
2    return print(1,to_string( empty ? 5 : 6)+"\n") -> 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(1,to_string( -33)+"\n") -> 0;
4  }
```

136

test-unary-negation.xtnd - Expected Output

```
1  −33
```

test-unary-negation-empty.xtnd

```
1  main([1,n] args){
2     return print(1,to_string( −empty)+"\n") −> 0;
3  }
```

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

```
1  empty
```

9. Git Logs

```
 1  23c2ae6 2016-12-17T13:00:05-05:00 GitHub: Merge pull request #123 from ExtendLang/size
       -asserts
 2  4c51203 2016-12-17T11:59:30-05:00 oracleofnj: Right confusion
 3  c05cf61 2016-12-17T11:52:34-05:00 oracleofnj: Fix import dir bug
 4  39edbb4 2016-12-17T11:46:06-05:00 oracleofnj: Merge branch 'master' into size-asserts
 5  339cb1f 2016-12-17T11:45:54-05:00 oracleofnj: Fix merge conflict
 6  7462381 2016-12-17T11:44:50-05:00 GitHub: Merge pull request #122 from ExtendLang/
       split-stdlib
 7  61ac8f2 2016-12-17T11:38:19-05:00 oracleofnj: Size asserts
 8  606af9f 2016-12-17T11:22:36-05:00 oracleofnj: Transform asserts into more useful form;
       add calc of assert value to codegen
 9  ee4f369 2016-12-17T10:40:17-05:00 oracleofnj: Combine asserts into a single expression
10  0f0f1c8 2016-12-17T10:38:56-05:00 Nigel Schuster: Added right and left to stdlib
11  fa43425 2016-12-17T10:30:23-05:00 oracleofnj: Split stdlib
12  824c53c 2016-12-17T10:11:13-05:00 Nigel Schuster: Added toUpper and toLower
13  ec24177 2016-12-17T10:02:30-05:00 Nigel Schuster: Implemented to and from ASCII
14  ab2e8f8 2016-12-17T09:15:39-05:00 GitHub: Merge pull request #116 from ExtendLang/line
       -plus
15  5d1610b 2016-12-17T09:08:57-05:00 GitHub: Merge branch 'master' into line-plus
16  df3a827 2016-12-17T09:08:48-05:00 GitHub: Merge pull request #117 from ExtendLang/cmd-
       args
17  32a3487 2016-12-17T09:02:27-05:00 GitHub: Merge branch 'master' into cmd-args
18  a8f9d33 2016-12-17T09:00:23-05:00 Nigel Schuster: Args
19  bfccf0c 2016-12-17T08:58:08-05:00 Nigel Schuster: Cut down line count for plus
20  a6bc89a 2016-12-17T08:48:31-05:00 GitHub: Merge pull request #114 from ExtendLang/only
       -new-string
21  5c96b7f 2016-12-17T08:03:27-05:00 GitHub: Merge pull request #109 from ExtendLang/unop
       -bitnot
22  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
23  375bea7 2016-12-16T23:56:35-05:00 oracleofnj: Merge branch 'unop-bitnot' into remove-
       interpreter
24  fb1bd77 2016-12-16T23:54:43-05:00 oracleofnj: Clean up; remove interpreter; change
       DimInt to DimOneByOne
25  539dd75 2016-12-16T23:46:35-05:00 GitHub: Merge branch 'master' into unop-bitnot
26  5668e53 2016-12-16T23:43:57-05:00 Nigel Schuster: Using lrint instead of fptosi
27  45691eb 2016-12-16T23:35:38-05:00 GitHub: Merge pull request #111 from ExtendLang/
       global-semant
28  2cdfb8b 2016-12-16T23:33:26-05:00 GitHub: Merge branch 'master' into global-semant
29  c9500d9 2016-12-16T23:33:14-05:00 GitHub: Merge pull request #112 from ExtendLang/
       remove-function-signatures
30  0c24f54 2016-12-16T23:25:23-05:00 oracleofnj: Remove return signature from grammar and
       all test cases
31  e7f2864 2016-12-16T23:03:53-05:00 oracleofnj: Merge branch 'cleanup-1' into global-
```

```
           semant
32  567507e 2016—12—16T22:53:20—05:00 oracleofnj: Check globals; use same symbol_table
           function for semant and codegen
33  55d8185 2016—12—16T22:00:30—05:00 Nigel Schuster: Removed comments and unneccessary
           files
34  629042f 2016—12—16T21:37:07—05:00 GitHub: Merge branch 'master' into unop—bitnot
35  48b139a 2016—12—16T21:34:09—05:00 Nigel Schuster: Implemented unary bitnot
36  39b02cd 2016—12—16T21:27:22—05:00 oracleofnj: Merge branch 'master' into global—semant
37  28c0983 2016—12—16T21:27:05—05:00 oracleofnj: Remove leftover printf
38  dc182df 2016—12—16T21:09:00—05:00 GitHub: Merge pull request #105 from ExtendLang/rg—
           eq
39  8cdf5c4 2016—12—16T19:31:26—05:00 oracleofnj: Expand test cases for range equality
40  41a3ccc 2016—12—16T19:18:44—05:00 GitHub: Merge branch 'master' into rg—eq
41  8dbebc1 2016—12—16T19:18:15—05:00 GitHub: Merge pull request #104 from ExtendLang/
           prevent—overlapping—formulas
42  c1431b5 2016—12—16T18:55:07—05:00 Nigel Schuster: Implemented basic subrange
           comparison
43  546536e 2016—12—16T18:47:12—05:00 oracleofnj: Detect overlapping formulas and give
           runtime error if present
44  3562e1b 2016—12—16T18:45:12—05:00 oracleofnj: Merge branch 'sr—val—fix' into prevent—
           overlapping—formulas
45  8713fa0 2016—12—16T18:42:40—05:00 oracleofnj: Checking
46  77d80b9 2016—12—16T18:26:31—05:00 Nigel Schuster: Fixed check for subrange
47  962c744 2016—12—16T12:09:00—05:00 GitHub: Merge pull request #101 from ExtendLang/
           finishing—these—range—literals
48  f234e00 2016—12—16T00:21:06—05:00 oracleofnj: Merge branch 'more—stdlib—functions'
           into finishing—these—range—literals
49  c9246ce 2016—12—16T00:20:59—05:00 oracleofnj: testing testing
50  6914039 2016—12—16T00:14:09—05:00 oracleofnj: Third time's the charm
51  4617e44 2016—12—16T00:01:12—05:00 oracleofnj: It compiles now
52  1d8e290 2016—12—15T23:42:43—05:00 oracleofnj: Fingers crossed
53  c9d28d3 2016—12—15T21:50:01—05:00 oracleofnj: Move all initializations into their own
           function; only box strings once
54  1cfdd16 2016—12—15T18:47:30—05:00 oracleofnj: Merge branch 'master' into more—stdlib—
           functions
55  19c2beb 2016—12—15T18:40:12—05:00 oracleofnj: Try a couple more things out
56  845cb04 2016—12—15T18:33:07—05:00 GitHub: Merge pull request #96 from ExtendLang/
           ternary—fix
57  4bfb3bc 2016—12—15T18:23:00—05:00 oracleofnj: Merge branch 'ternary—fix' into more—
           stdlib—functions
58  ae55ca4 2016—12—15T18:21:58—05:00 oracleofnj: Define cell_row, cell_col
59  30a5db6 2016—12—15T18:19:56—05:00 oracleofnj: Merge branch 'ternary—fix' into more—
           stdlib—functions
60  b9f1f10 2016—12—15T18:17:53—05:00 oracleofnj: What is truth?
61  ac84c2f 2016—12—15T18:15:37—05:00 oracleofnj: Fix ternary to work properly with ranges
62  1f57d91 2016—12—15T17:03:26—05:00 oracleofnj: Look at this one
63  437ba46 2016—12—15T16:56:04—05:00 oracleofnj: Try this one
64  f0edf5b 2016—12—15T16:46:52—05:00 oracleofnj: Fixing bug
65  5ba31e6 2016—12—15T14:17:52—05:00 GitHub: Merge pull request #94 from ExtendLang/nan—
           inf
66  67c5739 2016—12—15T14:17:46—05:00 GitHub: Merge pull request #93 from ExtendLang/type—
           typeof
67  48a3d5c 2016—12—15T14:05:37—05:00 oracleofnj: Improve test case
68  8f08227 2016—12—15T13:58:46—05:00 oracleofnj: Add isNaN and isInfinite to stdlib
69  cbeec74 2016—12—15T13:30:31—05:00 oracleofnj: Rename token
70  9582228 2016—12—15T13:18:09—05:00 oracleofnj: Rename type to typeof
```

71 d1422c7 2016—12—15T10:42:19—05:00 GitHub: Merge pull request #92 from ExtendLang/
       compiler
72 66689bb 2016—12—15T09:08:56—05:00 Nigel Schuster: added working directory option,
       doing testing completely in tmp
73 a13ae93 2016—12—15T09:08:31—05:00 GitHub: Merge pull request #91 from ExtendLang/
       sizeof
74 a31add9 2016—12—15T09:08:13—05:00 GitHub: Merge pull request #90 from ExtendLang/
       subselect—C—side
75 2e67e06 2016—12—15T09:01:06—05:00 Nigel Schuster: Added option to specify compiler,
       using clang
76 c171450 2016—12—15T02:33:48—05:00 oracleofnj: SizeOf
77 c168044 2016—12—15T00:48:35—05:00 oracleofnj: Add row(), column() to codegen, add
       print_endline() to stdlib.xtnd
78 bf9426d 2016—12—15T00:27:13—05:00 oracleofnj: Print subrange
79 407ce41 2016—12—14T23:02:02—05:00 oracleofnj: Merge in subrange_string
80 756ea8e 2016—12—14T22:51:00—05:00 oracleofnj: Ranges
81 27a8e79 2016—12—14T22:16:13—05:00 oracleofnj: Resolve RHS slice
82 876d056 2016—12—14T22:02:56—05:00 oracleofnj: Resolve RHS index
83 b59e022 2016—12—14T21:46:00—05:00 Nigel Schuster: Added method to print subragne as
       string
84 a7d53a8 2016—12—14T19:55:38—05:00 oracleofnj: Merge branch 'master' into subselect—C—
       side
85 362e85b 2016—12—14T19:55:23—05:00 GitHub: Merge pull request #88 from ExtendLang/
       subselect
86 4912fa3 2016—12—14T19:40:10—05:00 oracleofnj: Add debug print info for slice
       structures
87 c1b33f4 2016—12—14T18:58:45—05:00 oracleofnj: Builder to end all builders
88 5d400c2 2016—12—14T18:55:06—05:00 oracleofnj: Add selection builders
89 29f6e28 2016—12—14T18:20:51—05:00 oracleofnj: Make additional infix operator for
       populating structure element
90 046d096 2016—12—14T17:49:19—05:00 oracleofnj: Set up RHS slice types
91 b25c2f5 2016—12—14T16:49:17—05:00 GitHub: Merge pull request #87 from ExtendLang/make—
       a—selection
92 7a12082 2016—12—14T16:43:38—05:00 oracleofnj: Move selection test cases back into
       inputs
93 e2c08d5 2016—12—14T16:31:00—05:00 oracleofnj: Make IDs work with deref_subrange
94 02f2f0c 2016—12—14T15:21:31—05:00 GitHub: Merge pull request #86 from ExtendLang/
       include—stdlib
95 8b0503f 2016—12—14T15:18:14—05:00 GitHub: Merge branch 'master' into include—stdlib
96 1f034a0 2016—12—14T15:17:52—05:00 GitHub: Merge pull request #84 from ExtendLang/math—
       linker
97 1e6dd91 2016—12—14T14:58:44—05:00 oracleofnj: Add expected output for slurp
98 ff1a5e3 2016—12—14T14:53:38—05:00 oracleofnj: Remove extend_ prefix from all sample
       code
99 81a2828 2016—12—14T14:48:38—05:00 oracleofnj: Automatically add extend_ prefix to
       external functions
100 dcc1ed3 2016—12—14T14:30:52—05:00 oracleofnj: Fix samples
101 9b2c28f 2016—12—14T12:39:45—05:00 oracleofnj: Include stdlib automatically
102 13650ce 2016—12—14T12:35:21—05:00 Nigel Schuster: Merge branch 'math—linker' of https
       ://github.com/ExtendLang/Extend into math—linker
103 2e0d90d 2016—12—14T12:35:06—05:00 Nigel Schuster: Merge branch 'math—linker' of https
       ://github.com/ExtendLang/Extend into math—linker
104 83c689e 2016—12—14T12:34:14—05:00 Nigel Schuster: Merge branch 'math—linker' of https
       ://github.com/ExtendLang/Extend into math—linker
105 127f600 2016—12—14T12:34:07—05:00 Nigel Schuster: Include sys/resources
106 b34d97a 2016—12—14T12:03:44—05:00 GitHub: Merge branch 'master' into math—linker

```
107  8297f33 2016—12—14T12:01:47—05:00 GitHub: Merge pull request #85 from ExtendLang/put—
         lt—back
108  6b0c74f 2016—12—14T11:33:45—05:00 Nigel Schuster: Include sys/resources
109  37470e9 2016—12—14T11:14:06—05:00 oracleofnj: Put back LT, comment out sys/time.h
110  6bde590 2016—12—14T11:12:16—05:00 Nigel Schuster: Increasing stack size
111  6acc621 2016—12—14T11:03:31—05:00 Nigel Schuster: Disabled linking math when creating
         an intermediate
112  d87b73c 2016—12—14T10:51:58—05:00 GitHub: Merge pull request #82 from ExtendLang/hard—
         to—repro—bug
113  d126e3c 2016—12—14T00:51:00—05:00 oracleofnj: Try with time.h instead of sys/time.h
114  a535612 2016—12—14T00:48:35—05:00 oracleofnj: Remove lrints
115  e844853 2016—12—14T00:34:37—05:00 oracleofnj: Initialize all variables and remove
         pointer math; bug appears fixed
116  4c1a421 2016—12—13T22:55:07—05:00 oracleofnj: Some formula is weird
117  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
118  879eaf3 2016—12—13T22:43:17—05:00 oracleofnj: Testing
119  37f5ce2 2016—12—13T22:42:40—05:00 GitHub: Merge pull request #83 from ExtendLang/
         rounding—for—read
120  a1cfc5a 2016—12—13T22:34:21—05:00 Nigel Schuster: Added rounding at several places
121  e20f7e4 2016—12—13T21:36:13—05:00 oracleofnj: Half the time it works
122  61bc9b6 2016—12—13T20:33:27—05:00 GitHub: Merge pull request #81 from ExtendLang/fix—
         em—all
123  4a810df 2016—12—13T19:34:29—05:00 Nigel Schuster: Corrected testcase outputs
124  ae5b8a8 2016—12—13T19:08:43—05:00 GitHub: Merge pull request #80 from ExtendLang/
         select
125  70b2704 2016—12—13T19:02:32—05:00 oracleofnj: No C99
126  15fd762 2016—12—13T18:42:21—05:00 oracleofnj: Merge branch 'master' into select
127  8e6e9ba 2016—12—13T18:42:05—05:00 GitHub: Merge pull request #78 from ExtendLang/unop—
         unary—minus
128  7a93885 2016—12—13T18:41:49—05:00 oracleofnj: Calculate all formula indices
129  07e63dc 2016—12—13T18:19:58—05:00 oracleofnj: Properly build instantiate var
130  1a29129 2016—12—13T17:24:16—05:00 oracleofnj: Replace bools with chars for
         compatibility between C and LLVM
131  12e78a3 2016—12—13T17:17:54—05:00 oracleofnj: Added debug output
132  a483282 2016—12—13T16:13:30—05:00 oracleofnj: Merge branch 'master' into unop—unary—
         minus
133  f8c9b43 2016—12—13T16:13:09—05:00 oracleofnj: Make TypeOf work
134  8146d04 2016—12—13T16:12:17—05:00 GitHub: Merge pull request #75 from ExtendLang/fix—
         more—tc
135  94afc93 2016—12—13T16:02:35—05:00 Nigel Schuster: Corrected expected TC
136  f6f8276 2016—12—13T16:00:59—05:00 Nigel Schuster: Fixed string.xtnd file
137  dcd5766 2016—12—13T15:44:38—05:00 GitHub: Merge pull request #74 from ExtendLang/fix—
         tc
138  bfe1c07 2016—12—13T15:39:45—05:00 oracleofnj: Merge branch 'master' into unop—unary—
         minus
139  d9abfc0 2016—12—13T15:38:38—05:00 GitHub: Merge branch 'master' into fix—tc
140  50ed49c 2016—12—13T15:38:04—05:00 oracleofnj: Merging in main
141  23328f1 2016—12—13T15:37:18—05:00 GitHub: Merge pull request #73 from ExtendLang/and—
         or—xor
142  324779a 2016—12—13T15:32:26—05:00 Nigel Schuster: Corrected expected value
143  fafe2e6 2016—12—13T15:29:21—05:00 Nigel Schuster: Fixed string tc
144  022f05c 2016—12—13T15:23:59—05:00 Nigel Schuster: Fixed testcase
145  b12fe37 2016—12—13T15:18:57—05:00 Nigel Schuster: Implemented and, or and xor
146  90cbaa0 2016—12—13T15:16:31—05:00 Nigel Schuster: Added left and right shift
147  571ee7e 2016—12—13T14:56:05—05:00 Nigel Schuster: Merge branch 'power' of https://
```

```
                 github.com/ExtendLang/Extend into power
148  aeab40d 2016—12—13T14:55:57—05:00 Nigel Schuster: Removed unneccessary level of
        indirection
149  e377567 2016—12—13T14:53:28—05:00 GitHub: Merge branch 'master' into power
150  6ad8512 2016—12—13T14:53:11—05:00 GitHub: Merge pull request #69 from ExtendLang/unop—
        unary—minus
151  71f395d 2016—12—13T14:46:27—05:00 Nigel Schuster: Power to the people of Extend
152  6a04209 2016—12—13T14:45:46—05:00 oracleofnj: Fix merge conflict
153  edb0ecc 2016—12—13T14:43:32—05:00 oracleofnj: Add unary minus
154  668a0eb 2016—12—13T14:37:19—05:00 GitHub: Merge pull request #68 from ExtendLang/mod—
        div
155  866b68f 2016—12—13T14:32:18—05:00 Nigel Schuster: Added modulo and division operation
156  46d5aa6 2016—12—13T14:26:35—05:00 oracleofnj: Merge branch 'master' into unop—typeof
157  84dfc33 2016—12—13T14:26:25—05:00 Nigel Schuster: Crunched some code
158  76210eb 2016—12—13T14:26:18—05:00 oracleofnj: Start on it
159  f4d5a81 2016—12—13T14:22:12—05:00 Nigel Schuster: Merge branch 'master' into
        simplification
160  f873242 2016—12—13T14:21:26—05:00 GitHub: Merge pull request #65 from ExtendLang/
        subtraction
161  fc94112 2016—12—13T14:20:35—05:00 Nigel Schuster: Added multiplication
162  6c26c2c 2016—12—13T14:19:07—05:00 GitHub: Merge branch 'master' into subtraction
163  4afd78e 2016—12—13T14:18:55—05:00 GitHub: Merge pull request #64 from ExtendLang/
        refactor—boolean—binops
164  d4d4388 2016—12—13T14:15:58—05:00 GitHub: Merge branch 'master' into refactor—boolean—
        binops
165  bd90241 2016—12—13T14:14:17—05:00 GitHub: Merge branch 'master' into subtraction
166  4042259 2016—12—13T14:13:09—05:00 Nigel Schuster: Added subtraction
167  663f399 2016—12—13T14:12:57—05:00 oracleofnj: Remove wildcard from BinOp pattern match
168  82a3db2 2016—12—13T14:11:31—05:00 Nigel Schuster: Merge branch 'master' into
        subtraction
169  1bf6bed 2016—12—13T14:09:47—05:00 oracleofnj: Add TransformedAway exception for LogAnd
         and LogOr
170  c7d4162 2016—12—13T14:02:13—05:00 GitHub: Merge pull request #63 from ExtendLang/more—
        binops
171  952778e 2016—12—13T14:01:54—05:00 oracleofnj: Change Lt, Lte in grammar; implement GTE
172  97821c8 2016—12—13T13:47:52—05:00 oracleofnj: GT
173  1e1f973 2016—12—13T13:44:36—05:00 Nigel Schuster: Subtraction
174  e0a883a 2016—12—13T13:37:57—05:00 oracleofnj: Remove NotEq from AST since != is parsed
         to UnOp(LogNot,BinOp(Eq,...))
175  cc40008 2016—12—13T12:49:33—05:00 GitHub: Merge pull request #60 from ExtendLang/
        addition2
176  7123ebc 2016—12—13T12:41:09—05:00 GitHub: Merge branch 'master' into addition2
177  a656f57 2016—12—13T12:38:12—05:00 GitHub: Merge pull request #61 from ExtendLang/debug
        —unop
178  eb134b3 2016—12—13T12:29:53—05:00 Nigel Schuster: Moved testcases
179  044c6bd 2016—12—13T12:29:07—05:00 Nigel Schuster: Fixed off by one error
180  a64cc15 2016—12—13T12:14:45—05:00 oracleofnj: Add Debug expr
181  59858a0 2016—12—13T11:33:12—05:00 oracleofnj: Whoops no space
182  0426f34 2016—12—13T11:30:26—05:00 oracleofnj: Add test case
183  49ffa86 2016—12—13T11:19:14—05:00 GitHub: Merge branch 'master' into addition2
184  81533f4 2016—12—13T11:13:44—05:00 GitHub: Merge pull request #59 from ExtendLang/equal
        —rights
185  3cdaa5a 2016—12—13T11:12:41—05:00 Nigel Schuster: String addition
186  64d1760 2016—12—13T11:04:55—05:00 oracleofnj: Wake up please, GitHub
187  840aeaf 2016—12—13T10:48:03—05:00 oracleofnj: Remove usage demonstration
188  61ff439 2016—12—13T03:26:35—05:00 oracleofnj: Add string equality and test cases
```

```
189  f3112e9 2016—12—13T01:57:10—05:00 oracleofnj: Reduce cut & paste
190  08ce677 2016—12—13T01:35:46—05:00 oracleofnj: Remove obsolete testing file
191  ae8a07e 2016—12—13T01:23:26—05:00 oracleofnj: Merge branch 'print_value_p' into equal—
        rights
192  6090713 2016—12—13T01:22:47—05:00 oracleofnj: Use correct printf specifier
193  862b38c 2016—12—13T01:19:14—05:00 oracleofnj: Merge branch 'print_value_p' into equal—
        rights
194  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)
195  50281b1 2016—12—13T00:47:28—05:00 oracleofnj: Numeric equality
196  0f76aa4 2016—12—12T22:30:15—05:00 oracleofnj: Remove print flags
197  200b8b6 2016—12—12T22:16:15—05:00 GitHub: Merge pull request #57 from ExtendLang/
        addition2
198  da7c543 2016—12—12T12:43:31—05:00 Nigel Schuster: Setting flag for addition
199  7e7276b 2016—12—12T12:37:35—05:00 Nigel Schuster: Merge branch 'master' into addition2
200  8834635 2016—12—12T10:18:51—05:00 GitHub: Merge pull request #55 from ExtendLang/
        runtime
201  53ae9e0 2016—12—12T10:06:24—05:00 GitHub: Merge branch 'master' into runtime
202  6ed303e 2016—12—12T09:43:57—05:00 GitHub: Merge pull request #56 from ExtendLang/
        truthy—fix
203  ae49ce6 2016—12—12T01:15:29—05:00 oracleofnj: Remove extra file
204  7fe6a22 2016—12—12T01:11:53—05:00 oracleofnj: Falsey fix
205  d1e196d 2016—12—12T00:23:13—05:00 Nigel Schuster: Extracted runtime into seperate file
206  ecc620e 2016—12—12T00:17:06—05:00 GitHub: Merge pull request #54 from ExtendLang/final
        —draft—for—real
207  4c8caa5 2016—12—12T00:09:16—05:00 GitHub: Merge branch 'master' into final—draft—for—
        real
208  04d3b57 2016—12—12T00:00:29—05:00 GitHub: Merge pull request #39 from ExtendLang/more—
        lrm—ed
209  39025b0 2016—12—11T23:59:18—05:00 Nigel Schuster: Fixed examples, made small
        corrections
210  a875b41 2016—12—11T23:51:30—05:00 GitHub: Merge pull request #53 from ExtendLang/
        truthy
211  616dd34 2016—12—11T23:15:54—05:00 oracleofnj: Merge branch 'master' into truthy
212  0fa8255 2016—12—11T23:14:42—05:00 oracleofnj: Apparently still needs some work
213  78584d7 2016—12—11T23:09:07—05:00 oracleofnj: Thanks a lot Travis
214  b5673d2 2016—12—11T22:51:52—05:00 oracleofnj: TERRRRRRRRR NARRRRRRR
        EEEEEEEEEEEEEEEEEEE
215  b81bc1b 2016—12—11T22:04:25—05:00 oracleofnj: Maybe Truthy
216  b95d14f 2016—12—11T21:02:28—05:00 GitHub: Merge pull request #50 from ExtendLang/
        builder—hotfix
217  6dea96f 2016—12—11T20:40:47—05:00 oracleofnj: So many builders
218  8aa125f 2016—12—11T20:15:52—05:00 Nigel Schuster: Made som rpgroess
219  2a905c7 2016—12—11T19:15:47—05:00 GitHub: Merge pull request #47 from ExtendLang/
        function—parameter
220  2bc6c85 2016—12—11T19:11:33—05:00 oracleofnj: Add combined test case
221  860a11b 2016—12—11T19:04:35—05:00 oracleofnj: Merge branch 'master' into function—
        parameter
222  8c3499e 2016—12—11T19:03:39—05:00 oracleofnj: Remove extraneous printlines
223  99418c0 2016—12—11T19:02:31—05:00 oracleofnj: Make function parameters work
224  6c00a72 2016—12—11T18:45:46—05:00 Nigel Schuster: Some progress
225  387559b 2016—12—11T18:39:00—05:00 oracleofnj: First attempt
226  18fc1be 2016—12—11T18:08:11—05:00 GitHub: Merge pull request #45 from ExtendLang/empty
227  f7e9be8 2016—12—11T16:30:05—05:00 GitHub: Merge branch 'master' into empty
228  f1dd8a5 2016—12—11T16:18:44—05:00 GitHub: Merge pull request #46 from ExtendLang/
```

```
            actually—make—global—scope
229  50366f4 2016—12—11T15:38:05—05:00 oracleofnj: Make sure locals are properly masking
            globals
230  046c7cc 2016—12—11T15:30:53—05:00 oracleofnj: Make globals work, fix bug
231  a844a46 2016—12—11T15:14:09—05:00 oracleofnj: So close
232  18db166 2016—12—11T15:05:42—05:00 GitHub: Merge branch 'master' into empty
233  67849f0 2016—12—11T15:01:52—05:00 oracleofnj: Make the global scope object
234  393d02c 2016—12—11T14:25:02—05:00 Nigel Schuster: Implemented empty, small flag
            setting fix
235  3c4681d 2016—12—11T13:31:12—05:00 GitHub: Merge pull request #44 from ExtendLang/float
            —display—hotfix
236  7be1001 2016—12—11T13:26:55—05:00 GitHub: Merge branch 'master' into float—display—
            hotfix
237  abcffd0 2016—12—11T13:19:05—05:00 GitHub: Merge pull request #42 from ExtendLang/
            encapsulate—build—scope
238  556da44 2016—12—11T13:18:15—05:00 oracleofnj: Floating point math hotfix
239  0ad195e 2016—12—11T12:42:42—05:00 oracleofnj: Merge branch 'master' into encapsulate—
            build—scope
240  9caf464 2016—12—11T12:41:40—05:00 oracleofnj: Encapsulate a little more of building
            the scope
241  d65aad4 2016—12—11T12:09:28—05:00 GitHub: Merge pull request #40 from ExtendLang/make—
            global—scope
242  0f5a6ba 2016—12—11T12:04:05—05:00 oracleofnj: Merge branch 'master' into make—global—
            scope
243  56b58d9 2016—12—11T12:01:28—05:00 oracleofnj: Encapsulate build_var_defns
244  f25e5b3 2016—12—11T11:43:19—05:00 oracleofnj: Only construct var_defns once
245  9cee2fc 2016—12—11T10:07:36—05:00 Nigel Schuster: Testcases (#38)
246  f3f4bef 2016—12—11T00:45:44—05:00 oracleofnj: Make global variable to hold vardefns
247  a0ed757 2016—12—10T23:31:38—05:00 Nigel Schuster: Edited explanation for row() and
            column()
248  7c50ef2 2016—12—10T23:27:07—05:00 Nigel Schuster: Added info for strings
249  738e41b 2016—12—10T23:24:20—05:00 Nigel Schuster: Added boolean example
250  5377fdf 2016—12—10T23:19:26—05:00 Nigel Schuster: Added arithmetic example
251  a8f4ad9 2016—12—10T21:28:18—05:00 oracleofnj: Isolate the part of building a scope for
             reuse with global variables
252  58f7a4d 2016—12—10T18:05:01—05:00 Nigel Schuster: Performing copy before returning, so
             that memory can be freed with alloca
253  c0e56aa 2016—12—10T17:07:00—05:00 GitHub: Merge pull request #37 from ExtendLang/
            dereference
254  a4b35df 2016—12—10T16:42:17—05:00 Nigel Schuster: Removed obsolete methods
255  cf08a8c 2016—12—10T16:36:20—05:00 GitHub: Merge branch 'master' into dereference
256  ef0e5e7 2016—12—10T16:36:03—05:00 GitHub: Merge pull request #36 from ExtendLang/comp—
            warn
257  0177dc2 2016—12—10T16:35:50—05:00 GitHub: Merge pull request #35 from ExtendLang/
            linker
258  127f99d 2016—12—10T16:35:41—05:00 GitHub: Merge pull request #34 from ExtendLang/rel—
            import
259  b2e881d 2016—12—10T16:35:31—05:00 GitHub: Merge pull request #33 from ExtendLang/ts—
            fix
260  ce833d4 2016—12—10T16:14:34—05:00 Nigel Schuster: Dereferencing 1x1 subrange
261  e259556 2016—12—10T13:53:12—05:00 Nigel Schuster: Removed nodefaultlibs directive
262  09c3961 2016—12—10T13:50:19—05:00 Nigel Schuster: Modified linker to work for travis
263  36d662a 2016—12—10T13:37:27—05:00 Nigel Schuster: Attempt to link math
264  2d4564a 2016—12—10T13:22:14—05:00 Nigel Schuster: Linking math library
265  38ba6e6 2016—12—10T13:18:39—05:00 Nigel Schuster: Suppressing compiler warnings
266  9deac9b 2016—12—10T13:06:39—05:00 Nigel Schuster: Modified compile script. Removed
```

```
          debug output
267   d35607b 2016—12—10T13:04:30—05:00 Nigel Schuster: Simpler testscript
268   d37dac2 2016—12—10T12:36:45—05:00 Nigel Schuster: Fixed duplicate import issue
269   31c26bc 2016—12—10T12:30:29—05:00 Nigel Schuster: Added cmd args to link file
270   a350720 2016—12—10T11:40:50—05:00 Nigel Schuster: Switched import style from root
          directory to relative path
271   90e39b0 2016—12—10T11:24:19—05:00 Nigel Schuster: Fixed issue in testscript that might
           report false results when it fails early
272   718ecd3 2016—12—10T03:09:18—05:00 oracleofnj: Some changes to LRM; add if(a,b,c)
273   6a8f836 2016—12—09T18:29:22—05:00 GitHub: Merge pull request #24 from ExtendLang/final
          —draft—lrm
274   fc886a9 2016—12—09T18:23:52—05:00 oracleofnj: Merge branch 'final—draft—lrm'
275   cda63cb 2016—12—09T18:23:24—05:00 oracleofnj: Fix merge conflict
276   eac9e77 2016—12—09T18:04:08—05:00 GitHub: Merge pull request #29 from ExtendLang/
          refactor
277   fe825f4 2016—12—09T17:55:39—05:00 oracleofnj: Compact last bit
278   b02dbbe 2016—12—09T17:49:00—05:00 oracleofnj: Give formula functions names
279   edd7aa4 2016—12—09T17:40:57—05:00 Nigel Schuster: Removed artifcats
280   9b49e20 2016—12—09T17:37:59—05:00 Nigel Schuster: Fixed I/O testcases
281   a4ad4b1 2016—12—09T17:18:13—05:00 Nigel Schuster: Merge
282   b07398b 2016—12—09T17:17:19—05:00 Nigel Schuster: Added macro for function definition
283   ed01567 2016—12—09T17:17:06—05:00 oracleofnj: Make sizeof not break tests
284   a0a7054 2016—12—09T17:01:20—05:00 oracleofnj: Use symbol table
285   56fd61b 2016—12—09T16:11:10—05:00 oracleofnj: Merge branch 'refactor' of https://
          github.com/ExtendLang/Extend into refactor
286   38aedba 2016—12—09T16:10:35—05:00 oracleofnj: Create symbol table
287   dfb702e 2016—12—09T16:01:08—05:00 Nigel Schuster: Converted more to value_p from
          subrange_p
288   e963186 2016—12—09T15:42:35—05:00 Nigel Schuster: Made example TC work
289   eb76234 2016—12—09T11:14:58—05:00 Nigel Schuster: Made Hello World work again
290   08aeb70 2016—12—09T02:13:09—05:00 oracleofnj: Done for the night
291   cb39114 2016—12—09T01:35:36—05:00 oracleofnj: More refactoring
292   7974bbd 2016—12—08T23:53:31—05:00 oracleofnj: Banish the term extern
293   49af972 2016—12—08T23:45:30—05:00 oracleofnj: Add a couple comments
294   0fbf461 2016—12—08T21:52:24—05:00 oracleofnj: Get my bearings
295   5ecb599 2016—12—08T19:47:51—05:00 Nigel Schuster: Added some documentation
296   65066fc 2016—12—08T12:18:57—05:00 Nigel Schuster: Added name display for variable
297   fb18949 2016—12—07T23:44:17—05:00 oracleofnj: Merge branch 'master' into final—draft—
          lrm
298   4aab3dc 2016—12—07T23:43:25—05:00 oracleofnj: Update PDF
299   ed44d27 2016—12—07T23:43:01—05:00 oracleofnj: Fix failing test cases
300   9354fa7 2016—12—07T23:06:36—05:00 oracleofnj: Final draft candidate
301   78649f4 2016—12—07T18:09:46—05:00 oracleofnj: Almost done
302   05ded19 2016—12—07T15:47:52—05:00 oracleofnj: More work
303   f985cc8 2016—12—07T12:14:59—05:00 Nigel Schuster: Merge branch 'finish—transformations
          ' into get—val—rev
304   4b58ce9 2016—12—07T12:13:23—05:00 Nigel Schuster: Tried to add more instructions
305   0722412 2016—12—07T11:32:11—05:00 oracleofnj: Working
306   099efe7 2016—12—07T10:48:35—05:00 Nigel Schuster: Making progress on evaluating
          dimensions
307   fa09df7 2016—12—07T09:51:23—05:00 Nigel Schuster: Finally it works
308   cbb0577 2016—12—07T02:35:06—05:00 oracleofnj: Still WIP
309   e3c9436 2016—12—07T00:44:22—05:00 oracleofnj: WIP
310   b265e74 2016—12—07T00:41:23—05:00 Nigel Schuster: test commit to look at
311   18bb182 2016—12—07T00:35:06—05:00 oracleofnj: Still work in progress
312   a4554c0 2016—12—06T23:14:32—05:00 Nigel Schuster: At least it compiles
```

```
313  3432484 2016—12—06T22:42:22—05:00 Nigel Schuster: Getting closer. Need to add var_defn
        wrapper in build_formula
314  05145ca 2016—12—06T21:10:11—05:00 Nigel Schuster: Minor fix
315  af69b92 2016—12—06T17:23:45—05:00 oracleofnj: More updates
316  a65c24e 2016—12—06T16:14:10—05:00 oracleofnj: Merge branch 'master' into finish—
        transformations
317  85a4ccb 2016—12—06T16:12:31—05:00 oracleofnj: LRM update part 1
318  174a7b8 2016—12—06T11:09:31—05:00 Nigel Schuster: Made partial progress on
        implementing variable instanciation and such
319  90fc58e 2016—12—05T22:14:41—05:00 GitHub: Merge pull request #23 from ExtendLang/read—
        empty
320  767851d 2016—12—05T16:18:17—05:00 Nigel Schuster: Finished C side implementation of
        getVal
321  6b837d4 2016—12—05T16:06:34—05:00 Nigel Schuster: Merge branch 'master' into get—val
322  04c2c65 2016—12—05T15:53:35—05:00 oracleofnj: Add slurp by passing 0 max bytes
323  d8cf316 2016—12—05T14:46:46—05:00 oracleofnj: Start handling empty
324  910bd01 2016—12—05T14:27:07—05:00 GitHub: Merge pull request #21 from ExtendLang/
        fileio
325  1ce7f83 2016—12—05T14:18:41—05:00 oracleofnj: Create patch file
326  88480fb 2016—12—05T13:36:28—05:00 GitHub: Merge branch 'master' into fileio
327  29d02d9 2016—12—05T13:34:27—05:00 oracleofnj: Fix merge conflict — keep expr_loc
328  52e7a8a 2016—12—05T13:32:54—05:00 GitHub: Merge pull request #22 from ExtendLang/rm—
        micro
329  bfa906b 2016—12—05T13:28:03—05:00 oracleofnj: Fix off—by—one bug
330  eb8dd71 2016—12—05T13:20:03—05:00 oracleofnj: Address issues
331  f1b11ee 2016—12—05T12:46:35—05:00 Nigel Schuster: Skeleton for get_val
332  e4e5e26 2016—12—05T09:25:17—05:00 Nigel Schuster: Removed microc reference
        implementation
333  270da2b 2016—12—05T02:40:59—05:00 GitHub: Merge branch 'master' into fileio
334  b928e98 2016—12—05T02:40:10—05:00 Ishaan: Remove bloat
335  894b511 2016—12—05T02:32:49—05:00 Ishaan: Added testcase
336  62b8e83 2016—12—05T02:30:16—05:00 Ishaan: Added fwrite implementation
337  77a23ae 2016—12—05T01:39:30—05:00 Ishaan: Added read
338  46e9b58 2016—12—05T00:07:16—05:00 Ishaan: Make refactoring changes and new helpers
339  a5b9066 2016—12—04T14:00:30—05:00 GitHub: Merge pull request #20 from ExtendLang/lhs—
        all—ids
340  35e9471 2016—12—04T13:38:44—05:00 oracleofnj: Put back Id(s) as it was
341  641d454 2016—12—04T13:36:36—05:00 oracleofnj: Always transform to ID on LHS, even for
        LitInts
342  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
343  f47f2ba 2016—12—04T10:30:44—05:00 oracleofnj: Add error handling to close() and add a
        couple test cases
344  e95a95a 2016—12—04T10:07:01—05:00 oracleofnj: Add assertSingleNumber and get_number to
        eliminate more copy & paste
345  543e720 2016—12—04T09:47:03—05:00 oracleofnj: Add new_number() to eliminate some copy
        and paste
346  d7f10c9 2016—12—04T02:31:03—05:00 Ishaan: Tentative drafts of fileio functions
347  7d81e43 2016—12—04T00:15:20—05:00 oracleofnj: add diagnostic prinfs
348  868d9a4 2016—12—03T23:46:01—05:00 Ishaan: Cleanup
349  aa1e014 2016—12—03T23:42:46—05:00 Ishaan: Add file pointer array
350  88d05de 2016—12—03T18:38:34—05:00 Ishaan: Working on fopen
351  36f5848 2016—12—03T14:07:39—05:00 oracleofnj: Merge branch 'master' into finish—
        transformations
352  2ae2b83 2016—12—03T14:06:40—05:00 GitHub: Merge pull request #15 from ExtendLang/
        stdlib—fun
```

```
353  7c78a23 2016—12—03T14:02:51—05:00 oracleofnj: Move test_fabs out of regression test
         suite
354  0a8055b 2016—12—03T13:48:19—05:00 oracleofnj: make test | grep REGRESSION
355  a24742b 2016—12—02T22:50:43—05:00 Kevin: Merged stdlib with master
356  5243c5a 2016—12—02T18:16:36—05:00 Kevin: Removed magic numbers and add fabs test
357  330bec3 2016—12—02T13:49:34—05:00 oracleofnj: Merge branch 'master' into finish—
         transformations
358  8a60995 2016—12—01T23:38:54—05:00 GitHub: Merge pull request #18 from ExtendLang/
         parser—error
359  f0d33e2 2016—12—01T23:18:39—05:00 oracleofnj: Move error handling
360  3b24c3a 2016—12—01T23:16:53—05:00 oracleofnj: Adjust test script
361  60a732f 2016—12—01T22:55:28—05:00 oracleofnj: Merge branch 'master' into parser—error
362  5dec6a2 2016—12—01T22:55:05—05:00 oracleofnj: Thank you Nigel!!!
363  96a3028 2016—12—01T22:19:21—05:00 GitHub: Merge pull request #16 from ExtendLang/fail—
         silent
364  6c3696c 2016—12—01T21:59:40—05:00 oracleofnj: Figure out why test is failing
365  7912d5a 2016—12—01T21:26:03—05:00 GitHub: Merge branch 'master' into fail—silent
366  9702e5b 2016—12—01T21:14:35—05:00 oracleofnj: Merge branch 'master' into finish—
         transformations
367  5bdd52c 2016—12—01T21:13:45—05:00 GitHub: Merge pull request #17 from ExtendLang/
         lexbuf—pos
368  8893255 2016—12—01T20:35:04—05:00 oracleofnj: Add a couple test cases
369  2868653 2016—12—01T20:23:01—05:00 oracleofnj: Use lexbuf.lex_curr_p to calculate
         position
370  8c7b6ce 2016—12—01T18:59:49—05:00 GitHub: Merge pull request #11 from ExtendLang/
         parse_error
371  2885ac7 2016—12—01T18:56:15—05:00 Ishaan: Added test case for string
372  047cfec 2016—12—01T18:42:04—05:00 oracleofnj: Add short circuiting test cases
373  6acd7f6 2016—12—01T18:31:33—05:00 oracleofnj: Merge remote—tracking branch 'origin/
         fail—silent' into finish—transformations
374  72360f4 2016—12—01T17:09:08—05:00 Nigel Schuster: Minified error output for outputs
         that have not passed yet
375  5762112 2016—12—01T16:04:06—05:00 oracleofnj: Get rid of wildcard pattern match in
         interpreter
376  a90a343 2016—12—01T15:59:40—05:00 oracleofnj: Merge branch 'master' into finish—
         transformations
377  85bc21d 2016—12—01T15:59:05—05:00 oracleofnj: Remove unnecessary file
378  81fe565 2016—12—01T15:58:40—05:00 oracleofnj: Finish range literals
379  e9fb1c2 2016—12—01T15:04:03—05:00 Ishaan: Added increment to string buffer and tests
380  eb7c1e8 2016—12—01T15:04:03—05:00 Ishaan: Add partial character indexing
381  df09aea 2016—12—01T15:04:03—05:00 Ishaan: Add expected parse testcase intermediate
382  712a710 2016—12—01T15:04:03—05:00 Ishaan: Added tentative scanner—level line number
383  bf4ee6c 2016—12—01T15:04:03—05:00 Ishaan: Added SyntaxError Exception at scan level
384  da41520 2016—12—01T14:54:21—05:00 oracleofnj: So close
385  7abb394 2016—12—01T14:07:58—05:00 GitHub: Merge pull request #14 from ExtendLang/
         sinner
386  e0b7fdb 2016—12—01T14:05:38—05:00 Nigel Schuster: Rename empty to new_val
387  2cabadc 2016—12—01T11:58:03—05:00 oracleofnj: Merge branch 'master' into finish—
         transformations
388  6ea8cff 2016—12—01T10:10:26—05:00 Nigel Schuster: Using define instead of magic
         numbers
389  cd7d261 2016—12—01T10:07:10—05:00 Nigel Schuster: Merge branch 'master' into sinner
390  13cd317 2016—12—01T10:06:25—05:00 GitHub: Merge pull request #13 from ExtendLang/
         value_p
391  cf36f70 2016—12—01T09:47:38—05:00 oracleofnj: Sample digits function
392  4eeed07 2016—12—01T01:02:56—05:00 Ishaan: Change print return type to empty
```

```
393  fa42f27 2016—12—01T00:41:47—05:00 Kevin: Fixed acos function
394  53d34ad 2016—12—01T00:29:32—05:00 Nigel Schuster: Moved double values type to numeric
395  f769c61 2016—12—01T00:18:07—05:00 Nigel Schuster: Merge branch 'sinner' into stdlib—
         fun
396  3986f38 2016—12—01T00:17:21—05:00 Nigel Schuster: Merge branch 'value_p' into sinner
397  5bd87f9 2016—12—01T00:14:45—05:00 Nigel Schuster: Explicitly declaring to link math
         library
398  4604545 2016—12—01T00:12:08—05:00 Nigel Schuster: Consistently using floats
399  38b9824 2016—11—30T23:46:14—05:00 Nigel Schuster: Merge branch 'value_p' into sinner
400  3303575 2016—11—30T23:45:25—05:00 Nigel Schuster: Explicitly declaring to link math
         library
401  31a74ec 2016—11—30T23:35:34—05:00 Nigel Schuster: Merge branch 'master' into value_p
402  7f0bc86 2016—11—30T23:04:34—05:00 Kevin: Finished remainder of stdlib
403  cd160df 2016—11—30T22:50:18—05:00 Kevin: Added more c functions to stdlib
404  e085977 2016—11—30T19:59:57—05:00 Nigel Schuster: Made sin function work
405  206ee5a 2016—11—30T19:07:28—05:00 Nigel Schuster: Moved all function signatures to
         value_p return value
406  effc20b 2016—11—30T18:45:52—05:00 GitHub: Merge pull request #12 from ExtendLang/easy—
         compile
407  3b6d7b7 2016—11—30T17:51:19—05:00 Nigel Schuster: Added script to compile and link
408  febcff8 2016—11—30T15:54:45—05:00 oracleofnj: Add oddball formula test case and try
         out theory for range literal
409  4a1ff4f 2016—11—30T14:54:05—05:00 oracleofnj: Finish reducing Ternary to
         ReducedTernary
410  8f0a981 2016—11—30T12:35:43—05:00 oracleofnj: Working on reducing ternaries
411  d3c5812 2016—11—30T02:39:58—05:00 oracleofnj: Finish desugaring switch
412  0a22713 2016—11—30T00:09:10—05:00 oracleofnj: Getting ready to ternarize switch
413  84f016a 2016—11—29T21:54:15—05:00 oracleofnj: Fix bug in switch() with default case
414  d331b7a 2016—11—29T17:33:41—05:00 oracleofnj: Give desugaring variables easier—to—read
          names for debugging purposes
415  36f8de5 2016—11—29T16:14:46—05:00 oracleofnj: Missed one
416  d96da34 2016—11—29T16:13:21—05:00 oracleofnj: Transform &&, || into ternary
         expressions to support proper short—circuit evaluation
417  3a8efbc 2016—11—28T23:05:28—05:00 GitHub: Merge pull request #9 from ExtendLang/func—
         calls
418  7a2af49 2016—11—28T20:33:53—05:00 Nigel Schuster: Removed another ocaml 4.3 dep
419  468e79f 2016—11—28T19:50:53—05:00 Nigel Schuster: Added ocaml 4.3 as dep for travis (
         hopefully this works)
420  a408761 2016—11—28T19:35:49—05:00 Nigel Schuster: Fixed String.equal
421  90c3caf 2016—11—27T22:52:14—05:00 Nigel Schuster: Fixed interpreter for now
422  a18da78 2016—11—27T22:42:27—05:00 Nigel Schuster: Added accidentally created file
423  5647312 2016—11—27T22:41:22—05:00 Nigel Schuster: Made extern function calls work
424  872aa8c 2016—11—27T13:52:44—05:00 Nigel Schuster: Merge branch 'func—calls' of https
         ://github.com/ExtendLang/Extend into func—calls
425  26ef1cc 2016—11—27T13:51:06—05:00 Nigel Schuster: Merging list of functions
426  877336f 2016—11—27T12:15:11—05:00 GitHub: Merge branch 'master' into func—calls
427  5b3edb0 2016—11—27T12:14:43—05:00 GitHub: Merge pull request #8 from ExtendLang/stdlib
         —template
428  374273f 2016—11—27T12:13:52—05:00 Nigel Schuster: Function calls work now
429  952aab8 2016—11—27T09:54:12—05:00 Nigel Schuster: Merge extern
430  ac6268f 2016—11—26T23:06:00—05:00 Nigel Schuster: Boxing ints, added unop sizeof,
         actually returning subrange not dummy object
431  ca07be3 2016—11—26T21:27:19—05:00 Nigel Schuster: Unboxing hello world to and from
         subrange
432  aef6c19 2016—11—26T16:55:48—05:00 Nigel Schuster: Made Hello World somewhat workable
433  cfb637e 2016—11—25T18:27:37—05:00 Nigel Schuster: Fixed faulty setup on call
```

```
434  ebf926a 2016—11—25T17:48:57—05:00 Nigel Schuster: Added template in C
435  554fbb2 2016—11—23T22:28:29—05:00 oracleofnj: Better error message for WrongNumberArgs
436  f09e40e 2016—11—23T12:47:39—05:00 oracleofnj: Make sequence work
437  053980b 2016—11—22T16:02:27—05:00 oracleofnj: Actually commit all the extern stuff
438  0e0fa23 2016—11—22T14:36:54—05:00 Nigel Schuster: Added extern in Ast
439  aac63be 2016—11—21T23:52:25—05:00 oracleofnj: Better duplicate definition checking
440  08e2d07 2016—11—21T23:29:28—05:00 oracleofnj: Check assertions before evaluating fn
         return expression
441  69fa332 2016—11—21T18:01:23—05:00 oracleofnj: Add size assertions
442  22541c4 2016—11—21T12:48:34—05:00 oracleofnj: Fix bug in Call()
443  9a1d24b 2016—11—21T12:39:41—05:00 oracleofnj: Working on crazy bug
444  a485cee 2016—11—20T22:13:46—05:00 oracleofnj: Add test case for foo([m, n] arg)
445  10afe9a 2016—11—20T22:07:17—05:00 oracleofnj: Expand function signature
446  325e9ba 2016—11—20T18:53:52—05:00 oracleofnj: Well, this is awkward
447  0a76dc9 2016—11—20T18:41:12—05:00 oracleofnj: Add check of return value
448  488e34e 2016—11—20T18:31:39—05:00 oracleofnj: Add sample #1
449  93eebc5 2016—11—20T18:27:23—05:00 oracleofnj: Add semantic checking to make sure
         functions and variables on RHS exist
450  881f164 2016—11—20T17:22:40—05:00 oracleofnj: Check RHS slice to ensure end > start,
         otherwise evaluate to empty
451  442ae91 2016—11—20T11:42:54—05:00 GitHub: Merge pull request #73 from Neitsch/
         interpreter—global
452  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
453  367bc2b 2016—11—20T00:33:17—05:00 GitHub: Merge pull request #72 from Neitsch/codegen—
         part—app—fix
454  bdca834 2016—11—20T00:31:04—05:00 GitHub: Merge branch 'master' into codegen—part—app—
         fix
455  e956238 2016—11—20T00:28:49—05:00 GitHub: Merge pull request #71 from Neitsch/tc—fixes
456  9b742d1 2016—11—20T00:24:39—05:00 Nigel Schuster: Fixed partial function application
         warning
457  32f2989 2016—11—20T00:20:51—05:00 GitHub: Merge branch 'master' into tc—fixes
458  f87cb94 2016—11—20T00:20:35—05:00 GitHub: Merge pull request #69 from Neitsch/
         regression—tests
459  842ee5a 2016—11—20T00:18:56—05:00 GitHub: Merge branch 'master' into regression—tests
460  6d73717 2016—11—19T23:55:35—05:00 GitHub: Merge pull request #66 from Neitsch/fix—test
         —cases
461  05f317a 2016—11—19T22:37:36—05:00 Nigel Schuster: Fixed output on TCs
462  aa1d974 2016—11—19T22:33:40—05:00 Nigel Schuster: Fixed expected value for ternary
463  ab7653a 2016—11—19T22:32:27—05:00 Nigel Schuster: Fixed import testcases
464  848066c 2016—11—19T22:24:55—05:00 Nigel Schuster: Moved testcase asset to asset folder
465  53c9206 2016—11—19T22:21:48—05:00 Nigel Schuster: Corrected use of global variable in
         test_globals
466  5fe74a8 2016—11—19T22:21:00—05:00 Nigel Schuster: Fixed expected output for
         test_access_column_cells
467  214ab9d 2016—11—19T22:10:33—05:00 Nigel Schuster: Merge
468  fb31505 2016—11—19T22:08:42—05:00 Nigel Schuster: Passing testcases are in separate
         directory. Output of stats
469  5e39ba7 2016—11—19T21:55:03—05:00 Nigel Schuster: Merge
470  25263fe 2016—11—19T21:51:31—05:00 Nigel Schuster: Removed travis from build, removed
         super verbose output
471  0554ad9 2016—11—19T21:42:28—05:00 Nigel Schuster: Using precise lli version
472  04e5c4a 2016—11—19T18:30:32—05:00 oracleofnj: Add more operators to interpreter
473  e4a190c 2016—11—19T17:14:04—05:00 oracleofnj: Add argument to main and remove
         _expected from filenames
474  7cd2b3a 2016—11—19T16:53:12—05:00 oracleofnj: Merge branch 'master' into fix—test—
```

```
           cases
475  d1fddfd 2016—11—19T16:52:48—05:00 oracleofnj: Merge branch 'fix—test—cases' of https
         ://github.com/Neitsch/plt into fix—test—cases
476  36f72a1 2016—11—19T16:49:34—05:00 GitHub: Merge pull request #67 from Neitsch/
         test_cases
477  c46c87b 2016—11—19T16:47:26—05:00 GitHub: Merge branch 'master' into test_cases
478  642ce76 2016—11—19T16:39:50—05:00 Kevin: Fixed helloworld bug
479  ac3d7fa 2016—11—19T16:10:53—05:00 Kevin: Added corresponding AST result for gcd
         function
480  7b6b79e 2016—11—19T14:31:39—05:00 GitHub: Merge branch 'master' into fix—test—cases
481  a9320f3 2016—11—19T14:29:51—05:00 oracleofnj: Merge branch 'master' into fix—test—
         cases
482  24a3625 2016—11—19T14:27:48—05:00 oracleofnj: Add switch tests
483  de262b4 2016—11—19T14:24:39—05:00 GitHub: Merge pull request #60 from Neitsch/box—args
484  75e3f71 2016—11—18T20:39:23—05:00 oracleofnj: Fix parsing errors in test cases
485  4e38757 2016—11—18T16:00:10—05:00 GitHub: Merge branch 'master' into box—args
486  7146dce 2016—11—18T15:59:54—05:00 GitHub: Merge pull request #64 from Neitsch/reorg—
         test
487  f483ac7 2016—11—18T14:10:32—05:00 Kevin: Updated print statement for each test
488  09cb42f 2016—11—18T14:07:39—05:00 oracleofnj: Fix parse difference
489  39634bb 2016—11—18T14:01:21—05:00 oracleofnj: Remove unnecessary files
490  d772725 2016—11—18T14:01:02—05:00 oracleofnj: Make inputs work with interpreter
491  f4456f8 2016—11—18T13:17:25—05:00 GitHub: Merge branch 'master' into test_cases
492  00aafb7 2016—11—18T13:16:08—05:00 Kevin: Renamed inputs folder
493  99db652 2016—11—18T12:51:40—05:00 Kevin: Renamed expected output extension and created
          input folder for test cases
494  2825ada 2016—11—18T12:51:33—05:00 Nigel Schuster: Added branch to build
495  aafabb2 2016—11—18T12:50:56—05:00 Nigel Schuster: Verbose output for travis debug
496  124d61e 2016—11—18T12:44:50—05:00 GitHub: Merge pull request #61 from Neitsch/reorg—
         test
497  82cf599 2016—11—18T12:34:57—05:00 oracleofnj: Modify test script to compare
         interpreter and compiler with expected
498  faecfa1 2016—11—18T01:48:44—05:00 oracleofnj: Fix merge conflict in box_args
499  41a81ce 2016—11—18T01:40:11—05:00 oracleofnj: Move argument boxing into a function
500  6f63e89 2016—11—18T00:48:07—05:00 GitHub: Merge pull request #59 from Neitsch/hello—
         hello
501  088dc45 2016—11—18T00:29:45—05:00 Nigel Schuster: Merge
502  012caaa 2016—11—18T00:12:40—05:00 GitHub: Merge pull request #58 from Neitsch/copy—
         argv
503  f84757b 2016—11—18T00:02:34—05:00 Nigel Schuster: Removed unneccessary files
504  18fbff1 2016—11—18T00:01:49—05:00 Nigel Schuster: Removed dummy arg reading, added
         printing to interpreter — helloworld TC passes
505  b866da3 2016—11—17T23:31:42—05:00 Nigel Schuster: Made hello world work
506  9463afa 2016—11—17T23:12:41—05:00 oracleofnj: Merge branch 'copy—argv' of https://
         github.com/Neitsch/plt into copy—argv
507  54858ab 2016—11—17T23:11:29—05:00 oracleofnj: Add => infix operator to cut down on all
          the build_struct_gep calls
508  bb11d6d 2016—11—17T23:10:24—05:00 GitHub: Merge branch 'master' into copy—argv
509  e123652 2016—11—17T22:28:12—05:00 oracleofnj: Add byte for zero
510  26a03b7 2016—11—17T22:24:17—05:00 oracleofnj: Add new_string function
511  b8028f9 2016—11—17T20:27:37—05:00 Kevin: Removed files from test folder
512  c85d9b7 2016—11—17T20:25:21—05:00 Kevin: Move testcases to testcases directory
513  f17c6b6 2016—11—17T20:21:38—05:00 Kevin Ye: Complete testcases for List/Range/Function
         /Expression with expected outputs
514  5e63cee 2016—11—17T17:40:31—05:00 GitHub: Merge pull request #54 from Neitsch/
         operation_tests
```

515	4a4a806 2016—11—17T17:19:13—05:00 GitHub: Merge branch 'master' into operation_tests
516	cafe20e 2016—11—17T17:19:11—05:00 GitHub: Merge pull request #52 from Neitsch/one—main —arg
517	4b28df2 2016—11—17T17:17:44—05:00 GitHub: Merge branch 'master' into operation_tests
518	b728e2e 2016—11—17T17:16:20—05:00 GitHub: Merge branch 'master' into one—main—arg
519	d43a87b 2016—11—17T17:15:28—05:00 GitHub: Merge pull request #55 from Neitsch/shell— fix
520	b1238a0 2016—11—17T17:08:56—05:00 Nigel Schuster: Shell is not my strength
521	a6cc0ea 2016—11—17T17:05:09—05:00 Nigel Schuster: Screw you bourne shell
522	51fbe67 2016—11—17T16:59:50—05:00 Nigel Schuster: Using bourne shell style redirection :
523	3255e1b 2016—11—17T16:38:53—05:00 Ishaan: Modify test suite specs
524	f0ab4d8 2016—11—17T16:38:53—05:00 Ishaan: Moved expected output text files to directory
525	06d330c 2016—11—17T16:38:53—05:00 Ishaan: 75% through operator cases
526	e490548 2016—11—17T15:50:35—05:00 GitHub: Merge branch 'master' into one—main—arg
527	a4cf367 2016—11—17T15:50:29—05:00 GitHub: Merge pull request #51 from Neitsch/test— script
528	79ee3de 2016—11—17T15:18:58—05:00 oracleofnj: Call main() with first argument <empty> in interpreter
529	c4f7437 2016—11—17T14:39:38—05:00 Nigel Schuster: Removed version specific lli
530	7b2236b 2016—11—17T14:35:55—05:00 Nigel Schuster: Fixed if no flag is given
531	e10f656 2016—11—17T14:24:20—05:00 Nigel Schuster: Outputting diff only if —p flag is given
532	2d29597 2016—11—17T14:19:30—05:00 Nigel Schuster: Added it as build target
533	7af929a 2016—11—17T14:12:19—05:00 GitHub: Merge pull request #50 from Neitsch/test— script
534	6ea43f6 2016—11—17T13:54:55—05:00 Nigel Schuster: Added more env variables to avoid copy paste
535	05f27a2 2016—11—17T12:45:11—05:00 Nigel Schuster: Made simple testscript
536	aca43c1 2016—11—17T11:08:11—05:00 Nigel Schuster: Removed accidentally added files
537	9228eac 2016—11—17T04:52:31—05:00 Kevin Ye: Test cases for List of Tests and Range/ Function/Expression Tests
538	7feb392 2016—11—17T00:28:53—05:00 GitHub: Merge pull request #48 from Neitsch/ testing_list
539	6e42afa 2016—11—17T00:27:13—05:00 GitHub: Merge branch 'master' into testing_list
540	e40734b 2016—11—16T23:25:01—05:00 Ishaan: Added more test scenarios
541	41ef578 2016—11—16T17:50:03—05:00 GitHub: Merge pull request #49 from Neitsch/consume— command—line—args
542	3cbf089 2016—11—16T17:45:58—05:00 oracleofnj: Fix merge conflict
543	1570836 2016—11—16T16:51:05—05:00 GitHub: Merge pull request #45 from Neitsch/doc
544	a8fbced 2016—11—16T16:38:49—05:00 Nigel Schuster: Fixed minor syntax error
545	c2f37c8 2016—11—16T16:30:43—05:00 Nigel Schuster: Merge
546	2fa73be 2016—11—16T16:05:37—05:00 oracleofnj: Set return code to length of argv[1]
547	bc21af6 2016—11—16T15:54:12—05:00 Ishaan: Added initial testing list
548	cd0d156 2016—11—16T15:50:39—05:00 oracleofnj: Start processing command line args
549	4a1fcac 2016—11—16T13:55:46—05:00 GitHub: Merge pull request #46 from Neitsch/number— type
550	f1b481e 2016—11—16T11:04:44—05:00 Nigel Schuster: Added number type that defaults to int
551	8944b9a 2016—11—16T00:19:33—05:00 GitHub: Merge pull request #44 from Neitsch/fix—arg
552	92fb7a3 2016—11—15T23:57:37—05:00 Nigel Schuster: Added a little documentation
553	bcbde36 2016—11—15T23:49:07—05:00 GitHub: Merge branch 'master' into fix—arg
554	fa1741a 2016—11—15T23:03:23—05:00 GitHub: Merge pull request #43 from Neitsch/more— llvm—gen—js
555	57b2162 2016—11—15T22:39:38—05:00 Nigel Schuster: Using subranges instead of ranges

```
          everywhere
556  9407677 2016—11—15T22:31:03—05:00 oracleofnj: Add hash table for common functions and
         add dereference—the—range
557  46e1fd5 2016—11—15T21:38:51—05:00 oracleofnj: Eliminate some copy & paste
558  660c049 2016—11—15T20:54:33—05:00 GitHub: Merge pull request #42 from Neitsch/llvm—gen
559  25b23cd 2016—11—15T17:23:54—05:00 Nigel Schuster: Fixed column retrieval for 1x1
560  3f02203 2016—11—15T17:17:02—05:00 Nigel Schuster: Fixed tests
561  26b8fcf 2016—11—15T17:15:08—05:00 Nigel Schuster: Merge
562  e347a87 2016—11—15T17:12:26—05:00 Nigel Schuster: Using more generic flag for values
563  aed28b3 2016—11—15T17:08:07—05:00 oracleofnj: Add is_subrange_1x1
564  cf5cbf0 2016—11—15T14:51:40—05:00 oracleofnj: Merge branch 'llvm—gen' of https://
         github.com/Neitsch/plt into llvm—gen
565  c71d469 2016—11—15T14:51:19—05:00 oracleofnj: Replace String.equal with =
566  4b34abd 2016—11—15T14:41:37—05:00 GitHub: Merge branch 'master' into llvm—gen
567  a80a6d0 2016—11—15T14:41:07—05:00 oracleofnj: Add compile option to main
568  8ad5a19 2016—11—15T14:33:40—05:00 GitHub: Merge pull request #40 from Neitsch/
         interpreter
569  3f0362a 2016—11—15T14:28:44—05:00 GitHub: Merge branch 'master' into interpreter
570  c0c95a2 2016—11—15T14:16:13—05:00 Nigel Schuster: Merge
571  d5f4024 2016—11—15T13:44:44—05:00 Nigel Schuster: Moved failing TCs
572  42fd9ef 2016—11—15T12:21:57—05:00 oracleofnj: Fix bug in import
573  9c567c9 2016—11—15T11:11:30—05:00 Nigel Schuster: Working on imports, fixed most
         testcases
574  aa61ac9 2016—11—15T09:31:42—05:00 Nigel Schuster: Allocating scope object
575  cf1ebf9 2016—11—13T23:09:30—05:00 oracleofnj: Rewrite main to take options; fix bug
         where import didn't know about first filename
576  5749538 2016—11—13T21:59:28—05:00 Nigel Schuster: Added main function
577  d6daff3 2016—11—13T20:26:14—05:00 GitHub: Merge pull request #41 from Neitsch/
         LRM_String_Update
578  0a5d484 2016—11—13T18:45:29—05:00 oracleofnj: Revert "Generating function header"
579  6afe599 2016—11—13T18:44:58—05:00 Ishaan Kolluri: Added changes relating to strings.
580  137d7e2 2016—11—13T18:39:33—05:00 oracleofnj: Merge branch 'interpreter' of https://
         github.com/Neitsch/plt into interpreter
581  118bfc5 2016—11—13T18:38:34—05:00 oracleofnj: Allow single slice on RHS; make hashtag
         work
582  e376270 2016—11—13T17:55:41—05:00 Nigel Schuster: Added type arguments for functions
583  5cfb519 2016—11—13T17:26:23—05:00 Nigel Schuster: Set more types up
584  bf1d8bb 2016—11—13T15:30:35—05:00 Nigel Schuster: Merge branch 'interpreter' of https
         ://github.com/Neitsch/plt into interpreter
585  f83a0bc 2016—11—13T15:30:28—05:00 Nigel Schuster: Generating function header
586  3addcc8 2016—11—13T14:38:11—05:00 oracleofnj: Make size(expr) an operator instead of
         built—in function
587  9a74e14 2016—11—13T14:22:44—05:00 oracleofnj: Changing size() to be an operator
588  d6d2eaa 2016—11—13T00:08:41—05:00 oracleofnj: Add closure to interpreter_variable
589  64fba82 2016—11—12T22:38:39—05:00 oracleofnj: Added bsearch to show logic bug
590  66ffdb1 2016—11—12T19:21:07—05:00 oracleofnj: Add alpha version of function calls
591  376b29a 2016—11—12T17:17:23—05:00 oracleofnj: Add string as value type
592  08c61ee 2016—11—12T17:14:47—05:00 oracleofnj: Clean up discrepancies
593  a18d5fc 2016—11—08T11:38:22—05:00 oracleofnj: Fix bug with x[—1]
594  962f812 2016—11—07T23:27:08—05:00 oracleofnj: Refactor scope for interpreter; resolve
         variables on demand; make selections work properly
595  47bbef1 2016—11—06T22:05:55—05:00 oracleofnj: Minor adjustments to interpreter to work
          with mapped AST
596  fddc6bc 2016—11—06T18:32:17—05:00 oracleofnj: Eliminate extraneous nulls in JSON
597  ffddb17 2016—11—06T18:15:40—05:00 oracleofnj: Turn statement and function lists into
         StringMaps
```

598 6810003 2016—11—05T19:47:57—04:00 oracleofnj: Fix pattern matching warning
599 7107a46 2016—11—05T18:01:34—04:00 oracleofnj: Add function to check range literals for
        legality at parse time
600 80b13d1 2016—11—05T15:13:10—04:00 oracleofnj: Handle selections better
601 6cbb009 2016—11—04T15:48:58—04:00 oracleofnj: Count to 1,000,000 using tail—recursive
        versions of List.map and cartesian product
602 9b2252d 2016—11—04T15:25:13—04:00 oracleofnj: Show enter and exit
603 3585e43 2016—11—04T02:21:38—04:00 oracleofnj: See how high it can count recursively
604 38cf541 2016—11—04T02:15:50—04:00 oracleofnj: Get the easy parts of the interpreter
        working
605 5d81d6e 2016—11—03T17:17:51—04:00 oracleofnj: Start working on interpreter
606 0078cee 2016—11—01T23:40:57—04:00 oracleofnj: Got a non—tail—recursive version of
        topological sort working
607 85df175 2016—11—01T15:39:10—04:00 oracleofnj: Irrelevant highlighting thing
608 84c719a 2016—11—01T14:39:49—04:00 oracleofnj: Rearrange nested functions
609 557dc4e 2016—11—01T13:50:52—04:00 oracleofnj: Add circular import test case
610 c476798 2016—11—01T13:35:46—04:00 oracleofnj: Fix syntax errors
611 af5a31d 2016—11—01T13:31:49—04:00 GitHub: Merge pull request #37 from Neitsch/import—
        rec
612 d451cc4 2016—11—01T13:31:33—04:00 GitHub: Merge pull request #38 from Neitsch/import—
        load
613 02ca24f 2016—11—01T13:30:47—04:00 GitHub: Merge pull request #39 from Neitsch/wild—exc
614 6fa0e39 2016—10—31T16:43:17—04:00 Neitsch: Raising exceptions on certain values
615 e673dca 2016—10—31T15:56:43—04:00 Neitsch: Loading data from all imports
616 6a28c05 2016—10—31T15:40:41—04:00 Neitsch: Recursively looking up dependencies
617 3f28289 2016—10—31T11:53:10—04:00 GitHub: Merge pull request #36 from Neitsch/import—
        arrange
618 4eaef3b 2016—10—31T11:01:00—04:00 Neitsch: Removed obsolete parts
619 7d7b1e5 2016—10—31T10:59:12—04:00 Neitsch: Added unsorted function, globals and
        imports
620 7d70af2 2016—10—30T15:23:04—04:00 oracleofnj: Add some explanatory comments
621 40d6b16 2016—10—30T15:03:32—04:00 oracleofnj: More expansion samples
622 af9b01c 2016—10—30T14:48:44—04:00 oracleofnj: Refactor expansion code
623 903bc3f 2016—10—30T00:19:10—04:00 oracleofnj: Add test output
624 68b7b03 2016—10—30T00:17:02—04:00 oracleofnj: Add test case
625 a8bdf33 2016—10—30T00:04:05—04:00 oracleofnj: Add LHS slice expansion
626 4ee6fdf 2016—10—29T17:36:17—04:00 oracleofnj: Add output
627 2b8bced 2016—10—29T17:27:22—04:00 oracleofnj: Expand dimension expressions
628 443a818 2016—10—26T16:31:51—04:00 GitHub: Merge pull request #35 from ishaankolluri/
        master
629 9ba3c65 2016—10—26T16:31:00—04:00 Ishaan Kolluri: Add UNIs
630 022e8cd 2016—10—26T16:25:57—04:00 GitHub: Merge pull request #34 from ishaankolluri/
        master
631 808aae5 2016—10—26T16:22:10—04:00 Ishaan Kolluri: Added change to precedence operators
632 0bd9c4a 2016—10—26T15:59:53—04:00 GitHub: Merge pull request #33 from Neitsch/final—
        slicing—comments
633 fb2b382 2016—10—26T15:54:11—04:00 oracleofnj: Thats all for now folks
634 e7020ec 2016—10—26T15:00:11—04:00 GitHub: Merge pull request #32 from Neitsch/final—
        lrm—edits
635 4683f14 2016—10—26T14:48:41—04:00 oracleofnj: Flesh out switch expressions, add
        precedence
636 4b7984a 2016—10—26T11:15:03—04:00 GitHub: Merge pull request #31 from Neitsch/more—lrm
        —edits
637 3d587c5 2016—10—26T11:10:15—04:00 oracleofnj: Incorporate requested edits and a few
        more clarifications
638 0c42b9c 2016—10—26T09:22:08—04:00 GitHub: Merge pull request #30 from ishaankolluri/

```
      LRM_update
639   cd81040 2016—10—26T03:30:20—04:00 ishaankolluri: Added changes to first half of LRM
640   63fb02b 2016—10—26T02:13:17—04:00 GitHub: Merge pull request #29 from Neitsch/lrm—
         edits
641   0941e96 2016—10—26T02:04:47—04:00 oracleofnj: Rebuild PDF
642   cb04069 2016—10—26T02:04:01—04:00 oracleofnj: Add built in functions
643   4abf638 2016—10—26T01:56:38—04:00 oracleofnj: Add built in functions
644   7661925 2016—10—26T00:04:22—04:00 oracleofnj: Initial comments
645   5932551 2016—10—25T21:30:40—04:00 GitHub: Merge pull request #28 from Neitsch/func—doc
         —fix
646   cc66297 2016—10—25T20:14:27—04:00 Nigel Schuster: Fixed mistakes in functions part of
         the doc
647   b978f00 2016—10—25T13:04:05—04:00 GitHub: Merge pull request #27 from ishaankolluri/
         master
648   125a5bb 2016—10—25T12:49:38—04:00 Ishaan Kolluri: Removed AUX file
649   2e1ea60 2016—10—25T11:30:35—04:00 GitHub: Merge pull request #26 from Neitsch/better—
         regexp
650   84b03ee 2016—10—25T01:22:31—04:00 oracleofnj: Fix let order
651   91b40c5 2016—10—25T01:14:43—04:00 oracleofnj: Improve regexp
652   eb24036 2016—10—24T23:55:38—04:00 GitHub: Merge pull request #23 from Neitsch/file—io
653   991c918 2016—10—24T23:20:12—04:00 oracleofnj: Replace fopen, fclose etc. with open,
         close etc.
654   338faa0 2016—10—24T23:14:30—04:00 oracleofnj: Fix file inclusion and rebuild PDF
655   b24edd3 2016—10—24T23:11:50—04:00 oracleofnj: Merge in expressions section
656   44a1cc5 2016—10—24T23:06:07—04:00 oracleofnj: Merge scanner changes and add regexp to
         properly escape strings
657   2f09a64 2016—10—24T15:52:10—04:00 Kevin: Added the Expression Section 4 to LRM
658   1ea3c28 2016—10—24T15:26:16—04:00 oracleofnj: Merge branch 'master' into file—io
659   ec7cc9c 2016—10—24T15:21:23—04:00 Jared Samet: Replace repetitive code with more
         idiomatic OCaml
660   8cd39ac 2016—10—24T11:05:33—04:00 Kevin: Added string literals to scanner
661   e5d2478 2016—10—24T11:00:39—04:00 Kevin: Added string literals to scanner
662   a692466 2016—10—24T01:09:21—04:00 oracleofnj: Fix tests until strings ready
663   8553a50 2016—10—24T01:08:29—04:00 oracleofnj: Fix tests until string ready
664   0ed4ad7 2016—10—24T00:55:08—04:00 oracleofnj: Add File IO, Entry point and Example to
         LRM
665   71e0b1c 2016—10—23T22:58:21—04:00 oracleofnj: Fix section reference
666   92ac506 2016—10—23T22:39:06—04:00 Ishaan Kolluri: Make small change to data type
         section
667   6abb290 2016—10—23T22:34:42—04:00 oracleofnj: Initial commit for File I/O section
668   67b4b65 2016—10—23T19:30:03—04:00 Nigel Schuster: Reduce eye pain
669   2824ee9 2016—10—23T19:03:24—04:00 GitHub: Merge pull request #20 from Neitsch/samples
670   f8ae543 2016—10—23T18:23:11—04:00 GitHub: Merge branch 'master' into samples
671   13d0896 2016—10—23T18:20:03—04:00 GitHub: Merge pull request #19 from Neitsch/sequence
         —operator
672   e0c702d 2016—10—23T18:17:58—04:00 Neitsch: Fixed .gitignore
673   3a2cd60 2016—10—23T18:16:35—04:00 GitHub: Merge branch 'master' into sequence—operator
674   e42fe94 2016—10—23T18:05:48—04:00 Neitsch: Added code in LRM to test code samples
675   9d2cd17 2016—10—23T17:24:15—04:00 Neitsch: Merge branch 'master' into samples
676   167ddd2 2016—10—23T17:18:35—04:00 Neitsch: Removed test output
677   57319c4 2016—10—23T17:11:13—04:00 oracleofnj: Remove intermediate files
678   53824ea 2016—10—23T17:10:39—04:00 oracleofnj: Flip precedence of —> and ?: (?: is now
         lowest)
679   7dedf93 2016—10—23T17:05:23—04:00 oracleofnj: Add sequence operator to scanner/parser/
         AST
680   9805753 2016—10—23T17:01:31—04:00 GitHub: Merge pull request #17 from Neitsch/make—
```

```
               correction
681 e0c7aed 2016—10—23T16:59:33—04:00 Neitsch: Fixed test
682 ec3d682 2016—10—23T16:41:00—04:00 GitHub: Merge branch 'master' into make—correction
683 ea05658 2016—10—23T16:40:24—04:00 Neitsch: Moved sequence file
684 0ca56a0 2016—10—23T16:10:14—04:00 Neitsch: Merge
685 9d1094e 2016—10—23T16:08:59—04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
               config
686 0a28413 2016—10—23T16:08:59—04:00 Neitsch: Completed initial functions section doc
687 0797f32 2016—10—23T16:08:12—04:00 Neitsch: Changed subsection header
688 9df31f7 2016—10—23T16:08:12—04:00 Neitsch: Added dimension section
689 8939903 2016—10—23T16:07:26—04:00 Neitsch: Started working on Functions
690 cae3b37 2016—10—23T16:06:27—04:00 Neitsch: Added dimension section
691 049c95d 2016—10—23T16:06:08—04:00 Neitsch: Started working on Functions
692 84d20b5 2016—10—23T16:01:00—04:00 Neitsch: Comparing sample code with correctly parsed
                code in samples_comp
693 3f015ee 2016—10—23T15:52:01—04:00 GitHub: Merge pull request #18 from Neitsch/grammar—
               bug—fixes
694 7e558c1 2016—10—23T15:44:20—04:00 GitHub: Merge branch 'master' into make—correction
695 edf3dea 2016—10—23T15:44:20—04:00 GitHub: Merge branch 'master' into grammar—bug—fixes
696 d4961eb 2016—10—23T15:43:16—04:00 GitHub: Merge pull request #15 from Neitsch/
               functions—doc
697 0e0bda5 2016—10—23T15:05:42—04:00 GitHub: Merge branch 'master' into functions—doc
698 4652c67 2016—10—23T15:00:35—04:00 Neitsch: Added simple TCs, Moved Makefile to oasis
               config
699 b45718d 2016—10—23T02:27:36—04:00 oracleofnj: Modify grammar to allow [m,n] foo, bar,
               baz;
700 143fcba 2016—10—22T23:23:10—04:00 GitHub: Merge pull request #16 from Neitsch/more—AST
701 a726236 2016—10—22T20:51:27—04:00 oracleofnj: Add comments and sample program
702 8db4098 2016—10—22T19:44:48—04:00 oracleofnj: Fix minor grammar bug
703 80754c3 2016—10—22T18:19:27—04:00 oracleofnj: Hook up scanner and parser
704 660de8c 2016—10—22T13:54:32—04:00 GitHub: Add stuff to the grammar, minor corrections
               (#14)
705 cfe827d 2016—10—21T20:50:51—04:00 Nigel Schuster: Completed initial functions section
               doc
706 3609366 2016—10—20T21:14:00—04:00 GitHub: Update scanner.mll
707 0d57652 2016—10—20T21:10:27—04:00 Kevin: Fixed bug in scanner
708 1848813 2016—10—20T20:21:49—04:00 Kevin: Made scanner
709 1b610ac 2016—10—20T13:50:22—04:00 Nigel Schuster: Merge
710 acb9b93 2016—10—20T13:44:06—04:00 Nigel Schuster: Changed subsection header
711 b95d039 2016—10—20T13:43:51—04:00 Nigel Schuster: Added dimension section
712 71b93bb 2016—10—20T13:43:09—04:00 Nigel Schuster: Started working on Functions
713 a15772c 2016—10—20T13:38:08—04:00 GitHub: Merge pull request #10 from ishaankolluri/
               LRM
714 dee63c7 2016—10—20T13:26:28—04:00 GitHub: Merge pull request #1 from Neitsch/grammar—
               doc
715 dc93dbf 2016—10—20T13:18:29—04:00 Nigel Schuster: Grammar import
716 4d763cb 2016—10—20T12:44:52—04:00 Ishaan Kolluri: Made refactor and edits to intro
               section of LRM
717 e7443cc 2016—10—20T11:46:54—04:00 Ishaan Kolluri: Merging
718 7542b5d 2016—10—20T11:16:35—04:00 Nigel Schuster: Added dimension section
719 995cf83 2016—10—19T12:28:09—04:00 Nigel Schuster: Started working on Functions
720 40c2a5a 2016—10—19T03:43:06—04:00 ishaankolluri: Initial LRM Commit part 1
721 02a5c17 2016—10—18T18:38:21—04:00 Ishaan Kolluri: Added LRM initial info
722 d8794e9 2016—10—17T19:47:42—04:00 GitHub: Merge pull request #9 from Neitsch/
               documentation
723 70aa1b9 2016—10—16T13:36:23—04:00 Nigel Schuster: Added PDF Latex template
```

```
724  5111202 2016—10—14T19:59:45—04:00 GitHub: Added a bunch of stuff to the grammar: (#8)
725  da967e4 2016—10—12T13:24:50—04:00 Jared Samet: CFG Grammar (#6)
726  fea4e4b 2016—10—08T11:42:39—04:00 GitHub: There is no need to constantly build all
        branches. (#2)
727  7a5ccfc 2016—10—08T11:31:31—04:00 Nigel Schuster: Added greeting and newlines (#4)
728  10b17f7 2016—10—08T11:31:08—04:00 GitHub: Imported microc (#5)
729  726456f 2016—09—20T09:45:07—04:00 Nigel Schuster: [test] Add sample greeting to repo
        (#3)
730  9a2183d 2016—09—15T18:44:00—04:00 Nigel Schuster: Added merlin config
731  163e176 2016—09—14T18:51:53—04:00 Nigel Schuster: Moved whole build to script
732  d401eea 2016—09—14T18:43:58—04:00 Nigel Schuster: Added oasis opam package
733  ba7fd9c 2016—09—14T18:38:58—04:00 Nigel Schuster: Added ocaml configure (maybe this
        helps travis)
734  a461eae 2016—09—14T18:26:10—04:00 Nigel Schuster: Configuring opam environment for
        travis
735  ba2df2f 2016—09—14T18:19:26—04:00 Nigel Schuster: Added ocaml native compiler to apt
        package list
736  a8e5958 2016—09—14T17:24:36—04:00 Nigel Schuster: Added some more (possibly necessary
        opam packages
737  c54f5e3 2016—09—14T17:18:32—04:00 Nigel Schuster: Missed opam option
738  b10adf0 2016—09—14T17:13:57—04:00 Nigel Schuster: Fixed opam install
739  124f7f3 2016—09—14T17:08:09—04:00 Nigel Schuster: Fixed YML error
740  4909fa8 2016—09—14T17:03:54—04:00 Nigel Schuster: Using avsm source
741  4b24046 2016—09—14T16:58:33—04:00 Nigel Schuster: Allow sudo
742  e7b50db 2016—09—14T16:56:57—04:00 Nigel Schuster: Fixed setup order
743  f6d7ac4 2016—09—14T16:50:02—04:00 Nigel Schuster: Manually installing apt packages
744  f4084ab 2016—09—14T16:40:55—04:00 Nigel Schuster: Test commit
745  d7c5e9a 2016—09—14T13:15:43—04:00 Nigel Schuster: Initial commit
```