Sheets

A High-Level Programming Language for writing GPU Accelerated Applications

Gabriel Blanco (gab2135)
Amelia Brunner (arb2196)
Ben Barg (bbb2123)
Ruchir Khaitan (rk2660)

Table of Contents

1. Introduction

2. Language Tutorial

Simple Example

3. Language Manual

Lexical Conventions
- Identifiers
- Comments
- Whitespace

Types
- Fixed Point Type
- Floating Point Type
- Vector Types

Operators
- Arithmetic Operators
- Relational Operators
- Assignment Operator
- Vector Access Operator
- Operator Precedence

Standard Functions
- Declaration
- Function Calls
- Return Statements

Gfunctions
- Declaration
- Block
- Gfunction Calls
- Restrictions

Program Structure
- If Else Statements
- While Statements
- For Statements
- Scope

4. Project Plan

Process
- Planning
- Testing

Style Guide

Timeline

Roles and Responsibilities
Software Development Environment
Project Log
5. Architectural Design
Block Diagram
Preprocessor
Scanner
Parser and AST
Symbol Table
Semantic Checking and Code Generation
6. Test Plan
Source to Target
Test Suites
Test Automation
7. Lessons Learned
Gabriel Blanco
Amelia Brunner
Ruchir Khaitan
Ben Barg
8. Appendix
Preprocessor
Scanner
Parser
AST
Generator
Generator Utilities
Environment Types
Testing Script

1. Introduction

The *Graphics Processing Unit* (GPU) was invented in 1999 as a single-chip processor that allowed the CPU to offload graphics-intensive tasks to a separate processor. Unlike the CPU, which is built to contain only a handful of cores but a lot of cache memory, a GPU has limited memory but hundreds of cores, which, thanks to very efficient context switching, allows it to handle thousands of threads concurrently without significantly degrading the performance of the CPU. In the past, GPUs were seen as a luxury reserved for video processing and computer gaming. However, because of the advent of larger screen displays and a greater demand for video, image and signal processing applications, GPUs are quickly becoming more mainstream.

Although we are seeing more and more applications take advantage of the computational capabilities of the GPU, it is still very difficult to program for the GPU because of the many different types of GPU architectures and chip specific proprietary software. *Sheets* empowers application developers to take a high-level approach to programming on the GPU. With syntax, semantics and program structure to help programmers run parallelizable algorithms on the GPU, *Sheets* is a hardware-portable language that allows users to leverage the GPU's ability to handle large-vector data operations without needing to worry about hardware specifications. It does so by compiling down into OpenCL, an open-source programming language that can run on a wide variety of GPUs.

2. Language Tutorial

Sheets uses a very clean syntax that is syntactically similar to C, but with the use of white space delimiters and lack of semi-colons of a Pythonic program. The one requirement is every executable program must include a **`snuggle()`** function. The **`snuggle()`** is the first function that gets called by the CPU, and which determines the rest of the program execution. Confused about the name? Just remember: “What’s the first thing you do when you get into *Sheets*? You *snuggle*!”

2.1. Simple Example

Here is a very simple first program that simply adds to numbers, without any gfunc capabilities:

```
func int snuggle():
    int x = 1
    int y = 2

    int result = x * y
    return result
```

Now, let’s say that you want to perform this exact same operation, but over an array of 6 integers, and want to use GPU acceleration to speed up the calculation. This is how we would do it:

```
gfunc int[] gmultiply(int[] x, int[] y):
    for (int i=Block.start; i<Block.end; i=i+1;)
        Block.out[i] = x[i] * y[i]

func int[] snuggle():
    int[] x = [1.,2.,3.,4.,5.,6.]
```

```
    int[] y = [.5,.5,.5,.5,.5,.5]

    int[] result[6]
    return result = gmultiply(x,y)
```

Gfunctions (Functions that are executed on the GPU) have a unique syntax which abstracts away most of the complexities of performing operations on the GPU, while still enforcing conventions that make it possible for the functions to be called without violating the constraints of using a separate processing unit.

3. Language Manual

3.1. Lexical Conventions

3.1.1. Identifiers

An identifiers is a token given to a variable or function. They must begin with alphabetic character or underscore, followed by any number of alphanumeric characters or underscores. Identifiers are unique and cannot be double declared within the same scope, however you can mask a variable name within a higher level scope. Case matters, upper and lowercase letters are treated as distinct.

3.1.2. Comments

Sheets supports both inline and block comments, although there is no support for nested block comments. The comments in *Sheets* use the octothorpe, since they look a little bit like a threaded sheets:

```
## Inline Comment

#~ A Block Comment ~#

#~
 ~ A Multiline Block Comment
 ~#
```

3.1.3. Whitespace

Any whitespace found after a new line is syntactic, and used to determine scope. Whitespace is any space character, but *Sheets* does not allow the use of tab characters, since the size of a tab can vary in depending on the environment. Whitespace after the last non-whitespace, non-comment token is not syntactic and gets stripped away during compilation.

Sheets allows for statements that are more than one line long to the use of the line-join character \ as shown below. Note that the whitespace indentation on the following line is not syntactically important, since it is simply a continuation of the first line.

```
<some_part_of_statement> \
  <rest_of_statement>
```

3.2. Types

Sheets supports two kinds of primitives, both of which are numerical: *Fixed Point* and *Floating Point*.

3.2.1. Fixed Point Type

The fixed-point numerical primitive is the `int`. An `int` is a simply signed (two's complement) 32-bit integer type. Fixed-point literals can be declared as a sequence of numeric characters, although any number that is larger than the maximum integer size (`±2147483647`) will cause a compilation error.

3.2.2. Floating Point Type

The floating-point numerical primitive is the `float`. A `float` is a simply signed (two's complement) 32-bit single precision number, as defined by IEEE 754. Floating-point literals can be declared as a sequence of numeric characters followed by a decimal, and then an optional fractional component. If the decimal is omitted, the compiler will assume the literal to be a fixed-point type, which may lead to compilation errors.

3.2.3. Vector Types

Single Dimensional Arrays, or Vectors, represent multiple instances of either floating or fixed point numbers allocated in contiguous ranges of memory: either on the stack, the heap or in GPU memory (either globally or locally). Vectors have identical representations on both the CPU and the GPU, and *Sheets* automates data transfer between the CPU and GPU automatically. Vectors are zero indexed and can be accessed using square-bracket notation.

The syntax for declaring a new array is as follows; `int[]` is the type, `SIZE` is the integer number of elements allocated:

```
<vector_type> sampleVector [SIZE]
```

Vector literals can be defined within brackets, given that each element within is a literal of the singleton type of the vector. Declaring a vector literal with mixed types or larger than the allocated size will result in a compilation error

```
int[] sampleVector[4] = [1,2,3,4]
```

Two types of vectors are supported, fixed point `int[]` and floating point `float[]`.

3.3. Operators

These are all the operators supported in *Sheets*:

```
+   addition
-   subtraction
*   multiplication
/   division
==  equivalence
!=  non-equivalence
<   less-than
>   greater-than
<=  less-than-or-equal-to
>=  greater-than-or-equal-to
=   assignment
```

3.3.1. Arithmetic Operators

Sheets provides the four basic arithmetic operators: *addition*, *subtraction*, *multiplication* and *division*. All arithmetic operators are left-associative binary operators, and multiplication and division have higher operator precedence than addition and subtraction.

3.3.2. Relational Operators

The relational operators compare the values of two expressions, the operators are *equivalence*, *non-equivalence*, *less-than*, *greater-than*, *less-than-or-equal-to* and *greater-than-or-equal-to*. They are used within conditional statements, such as *if*, *else*, *while*, or *for* blocks to control execution of code by testing a boolean condition.

3.3.3. Assignment Operator

The *assignment* operator = is a right-associative binary operator that takes the value on the right hand side of the operator and stores it in the left hand side. Only certain types of expressions, known as *assignment-expressions*, are allowed to be on the left hand side, the "receiving" side of the operator. *Assignment-expressions* are limited to *variables* and *array elements*.

3.3.4. Vector Access Operator

The double brackets `[]` are used to denote a right associative access of the array label immediately to the right, where the expression within the brackets has to be of type int. The value returned by the array access operator is the *k*th element in the array where *k* is the integer that the expression within the brackets evaluates to.

```
int[] sampleVector[4] = [1,2,3,4]
sampleVector[0] = 2
```

3.3.5. Operator Precedence

From highest to lowest precedence, where highest order operators bind tighter than lower order ones:

- Multiplication and Division
- Addition and Subtraction
- Relational Operators

- Assignment

3.4. Standard Functions

On the global scope, you are only allowed to declare variables, so any operations on a variable needs to be done from within a function. Program execution is determined by the `snuggle()` function, the first function called during execution that does not take any arguments. All subsequent functions, both standard and gfunctional can then be called either from `snuggle()`, or from another declared standard function.

3.4.1. Declaration

Standard functions have this declaration syntax:

```
func <type> <name> (arg1, arg2, … ):
    ...
```

Where `<name>` is a unique identifier, and `<type>` is the type for the returning value. The arguments are optional variable declarations for arguments that get passed in during the invocation of a function. When a function is called, the *function call statement* must be have the same number, ordering and type as the variables declared in the declaration.

3.4.2. Function Calls

Functions are called by using the unique identifier with parentheses to pass in arguments. The syntax for a *function-call* expression is:

```
<name>(arg1, arg2, … )
```

Where the number of arguments (or lack thereof) must match what has previously been declared.

3.4.3. Return Statements

If a function contains a return statements, it can return a value of the type specified in the function declaration. If the value of a return statement does not match that of the function, the compiler will throw an error.

3.5. Gfunctions

Gfunctions are a special subset of functions that get executed on the GPU. *Gfunctions* have a special syntax in *Sheets* that allows programmers to write functions that work with the constraints of the GPU. When a *gfunction* is called, the body of a *gfunction* as well as a copy of the input parameters is sent to a group of threads on the GPU to be processed concurrently. This threaded processing allows for faster vector operations.

3.5.1. Declaration

Gfunctions are declared in much the same way as a standard function, although with the option to define a *Block Size*: the range of indices within a vector that the each instance of a *gfunction* has access to during parallel execution. The syntax for declaring a *gfunction* is:

```
gfunc <vector_type> <name>(arg1, arg2, ... ).[BLOCK_SIZE]:
    ...
```

The gfunction *type* must be a vector, since Gfunctions are intended to be used for large vector operations. The block size declarator at the end of the gfunction declaration is optional. If it is omitted, the `BLOCK_SIZE` will be set to 1.

3.5.2. Block

In the body of the gfunction, you have access to a *gfunction-specific* language construct called `Block`. `Block` stores several variables that can be accessed by using the dot operator. These are the variables:

`Block.size`

> The number of elements that an instance of a *gfunction* can access. This number is set by the programmer in the *gfunction declaration*. The purpose of defining a `Block.size` is that in order for a function to be parallelizable, the operations on a vector need to be able to be partitioned and performed in parallel. The `Block.size` defines the minimum number of vector elements in each group of that gets sent to a single thread on the GPU.

`Block.start`

> The starting index for an instance of the a gfunction. Every instance of a gfunction handles a different, non-overlapping section of the array so that no write conflicts occur.

`Block.end`

> The ending index for an instance of the a gfunction. The difference between the starting and ending index is always the block.size, so that no two instances of a gfunction handle overlapping sections of an array.

3.5.3. Gfunction Calls

```
<name> (arg1, arg2, ...)
```

Gfunctions are applied processed on the GPU, and program execution is sequential, so a *gfunction* call handles all of the GPU memory and thread management internally, and blocks until a value of vector type is returned from the *gfunction*

3.5.4. Restrictions

Since a GPU provides a tremendous boost in computational throughput by adhering to a Single Program Multiple Thread (SPMT) model wherein a single sequence of instructions are loaded onto many different cores and executed concurrently, any situation where a core has to halt execution to load other instructions, (for example, in a program with a lot of nested branching) leads to a significant increase in execution time. Therefore, GPU execution is primarily suited for a simple combinations of operations applied independently and concurrently to multiple disjoint parts of a data set, and to further reinforce best practices and maintain high performance, *gfunctions* can only access primitive operations and control flow structures and can not call other functions from within the body of a *gfunction*. Also, *gfunctions* cannot access variables declared in the program's global scope, as *gfunctions* operate within a separate GPU memory space that does not replicate variable's declared globally within the CPU memory. However, global variables can be passed as arguments to *gfunctions*.

3.6. Program Structure

The statements outside of any function definitions are in global scope. The only statements allowed in global scope are variable declarations for variables accessible by name from all functions, gfunction definitions and function definitions. The starting execution point for a Sheets program is the required snuggle() function that is required to be the last function definition in a Sheets file.

3.6.1. If Else Statements

Sheets supports *if* / *else* statements that allow for conditional branching of execution based on some boolean expression. An *if* / *else* statement introduces a new scope, and so the body of an *if* statement which executes if the boolean expression evaluates to true, has to be consistently indented. Conditional statements can nest freely, and finally, each *if* statement can optionally be paired with an *else* statement, which has to be at the same scope as the if statement and come immediately after the body of the matching *if* statement.

```
<func OR gfunc> <vector_type> <name>(arg1, arg2, ... ):
    …
    if(<boolean_expression>):
        …
    else:
        …
```

3.6.2. While Statements

```
<func OR gfunc> <vector_type> <name>(arg1, arg2, ... ):
    …
    while (boolean_expression>):
        …
```

Sheets supports *while* statements that repeatedly execute the code in the scoped block directly under them as long as the boolean expression within the while statement evaluates to *true*.

3.6.3. For Statements

```
<func OR gfunc> <vector_type> <name>(arg1, arg2, ... ):
    …
    for (<assignment_statement>; boolean_expression>; <assignment_statement>):
        …
```

Sheets supports *for* loop statement for repeated code execution as well, where the first assignment statement passed to the *for* statement is executed only once, and then the scoped block under the body of the *for* loop is executed as long as the boolean expression in the middle of the *for* loop evaluates to true. Each time, before the scoped block of statements executes, first, the boolean expression is reevaluated, and after each execution, the third assignment in the *for* statement is executed.

3.6.4. Scope

Variables declared in a function or gfunction are defined as being in function scope, and are visible or available to be referenced by name within the body of the function. Similarly, the body of a for, while, or if statement introduces a new subscope, where variables declared within that scope are only accessible to be referenced by name within that scope level.

Note that Sheets allows references to variables in a higher scope level (such as global scope) from within a lower sub-scope level and explicitly not the other way around. Also, variables declared in a lower scope level can mask variables declared in a higher scope level.

4. Project Plan

4.1. Process

4.1.1. Planning

Our team met to discuss the project twice a week, on Wednesdays with our T.A. Kuangya Zhai, and then again on Thursday as a team. With our weekly meetings with Kuangya, we would gage our progress and ask questions about any roadblocks we encountered during the week, and then in our weekly team meeting we would assign action items to try to have done by our next meeting. Of course, we did not always meet the goals set every single week, but by setting weekly goals we made sure that the project was always progressing.

There were often moments of “bottlenecking”, times where one member could not continue working on their part of the project until another member had finished their task. We found that the best way to avoid bottlenecking was to maintain good communication, so that if one member’s task was more difficult than previously thought, other members could offer support and help fix the issue.

4.1.2. Testing

We were only able to do full stack testing during the last week, since we only started generating code in the in the last two weeks of the project. For that reason, we developed test suites that tested individual components of the compiler. This meant that when writing a certain part of the compiler, one of the people working on that component was also expected to be writing tests that were expected to either pass or fail the specific module.

Even if the tests failed downstream in an unimplemented part of the compiler, analyzing the error message could informer the programmer whether the test was failing in the current module, upstream or downstream. We will cover the specifics of testing in more depth in Section 6.

4.2. Style Guide

The group used these conventions when writing code to ensure consistency and better readability:

- Lines of code should not be longer than 80 characters
- For consistency over different machines, use only spaces, no tabs, for indentation
- Each program uses either 2-space or 4-space indentation, do not mix within a program.
- Use under_scores for naming variables (as opposed to camelCase).

4.3. Timeline

September 23rd	First commit, creation of project repository
September 24th	Submitted *Project Proposal*
October 27th	Submitted *Language Reference Manual*
November 21st	Finished *Preprocessor* and *Scanner*
December 1st	Finished *Parser* and *AST*
December 8th	Successfully Generated Code
December 17th	Project presentation and submission of Final Project Report

4.4. Roles and Responsibilities

As the project progressed, the roles of each team members became a lot more fluid than the initially assigned roles of *Tester*, *System Architect*, *Project Manager* and *Language Guru*. Each member was heavily involved in the development of a least two of the major components of the Compiler, and worked to develop tests for of the system they were building and communicating with other members to make sure their component interfaced correctly with the other systems. The team frequently worked together in the same room, which means that a lot of the roles overlapped as we coded in together in pairs and exchanged roles.

Team Member	Responsibility
Amelia Brunner	Compiler Front End, Code Generation
Gabriel Blanco	Compiler Front End, Automated Testing Suite
Ruchir Khaitan	Compiler Back End, Semantic Checking
Ben Barg	Compiler Back End, GPU Environment Testing

4.5. Software Development Environment

Because we wanted to execute our language on a physical GPU, we targeted our build environment towards an Amazon Web Services EC2 instance with a GPU driver.

Our Stack:

- **Github-Hosted Git Repository** - for version control, which contains our compiler code, tests and also a small C library to encapsulate some of the OpenCL boilerplate
- **Python 2.7** - for a preprocessor that parses Sheets' *whitespace* scoping into a whitespace-independent intermediate representation that can be tokenized by an ocamllex-generated lexer.
- **OCaml 4.2.01** - for parsing and semantic checking of our preprocessed Sheets code and generation of C and accompanying OpenCL kernels.
- **GCC** - for building the C output of our compiler.
- **OpenCL 1.1** - headers and libraries for both the Nvidia GPU and Intel CPU, including the just-in-time compiler used by the C output of our compiler.

Our testing environment:

- *Amazon Web Services EC2 g2.2xLarge* Instance with the above packages installed
- *Nvidia Grid K520* GPU Driver Accessible by the AWS instance
- *Intel Xeon E5-2670* CPU Accessible by the AWS instance

4.6. Project Log

This project log shows a history of 463 commits starting from September 23rd and ending December 17th. As you can see, each member was heavily involved in the development of the project over a 3 month period.

```
commit 92345e70bbc057d6b9f2a8a776ae185e2e8a7f6c
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Wed Dec 17 13:36:38 2014 -0500
commit 693f4fd9ded8ae4d88e7ee3409135e865ee8934a
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Dec 17 11:54:18 2014 -0500
commit f57b42a567e824b9f718a64665c818d0753d0fa4
Author: Benjamin Barg <benbakerbarg@gmail.com>
```

```
Date:   Wed Dec 17 15:44:58 2014 +0000
commit ac2234e878a30bd416d14fa863b5f9eef1fc3adf
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 10:28:18 2014 -0500
commit 0911e87228148f3fa53a6aa36e91a74042abb940
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:45:53 2014 -0500
commit 08e5bdc43b6aa072627717e13ca415fc63cda803
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:33:12 2014 -0500
commit c3868e4817747c186455f73b292d03902c7d35bc
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:33:09 2014 -0500
commit 00fcd16ae573e12eabdcaa5fa1224d7491c433ff
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 07:24:33 2014 -0500
commit 3d2c30ec80ad6f68c883f2c8fd63e568405b1557
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 07:22:37 2014 -0500
commit 41949e38da786f7042a8ef6a46729b592084b7d1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:20:15 2014 -0500
commit 15c5d16e2df1d5367ea0f68f3f37ca6348d640a4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:17:18 2014 -0500
commit aa629b06d6c5624ba82d0e591ccc0f758b7aba7f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:13:50 2014 -0500
commit b81bb2adf2e1a3a66f26e99a97f4ebb2f01b5640
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 07:11:47 2014 -0500
commit 852a7008e4266752afa12c4c4076495d39736dc6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:51:19 2014 -0500
commit 6d8e0f4e8596c3b4eee5d3265ef23dbf3f45583d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:22:04 2014 -0500
commit d7c35dfeea319dae309e18fecf6f4f848e3bfcc4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:22:02 2014 -0500
commit ef560a8217fe06e17e5faa91f27add14928c2c82
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 06:20:00 2014 -0500
commit 0e4a720ca997866d6bcad05637c4bc421323b842
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 06:14:09 2014 -0500
commit 41981d16f21c4426d2d6d5792a8bdbb8202ccc63
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 06:14:05 2014 -0500
commit 8e619f024f556f1b19df8751b55b28039a6bfd89
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:12:08 2014 -0500
commit 0e4ba88e30ef635cbda0312553dc5b0c40e10031
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:10:23 2014 -0500
commit f6948503ec2db36bd276cc044f1966f179c19af9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 05:58:07 2014 -0500
commit ca3ace02ae8b52ffba7bd7995c8f074ceddef2a7
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 05:57:52 2014 -0500
commit e0d086e9a14fd2d69ab316957a68d7a0e81047cc
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 05:47:13 2014 -0500
commit 87a192fa902a101ceee7786d99db00ced9591693
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 05:29:41 2014 -0500
```

```
commit ae335a5c7d253e7791f2dde47d0e943fb3f6b086
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 05:19:21 2014 -0500
commit 2638c463ea62b4886a2c579e7089210db92db51d
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 04:52:37 2014 -0500
commit ddaf3229776fce304f368c537ec6a45ad2c48322
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 04:28:06 2014 -0500
commit 7eb8ff387d5c48e6e80740a81ac07c2d19fa1193
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 03:50:26 2014 -0500
commit 9f1db96f1c41169224cbf64df6fe0a7460cb8449
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Wed Dec 17 03:38:53 2014 -0500
commit f94b41e697aad065a725cf4df97428067641a39a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 03:10:14 2014 -0500
commit 586cdeb0c91f22f67a3280a89d4687b3458db1c2
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 02:06:43 2014 -0500
commit f69804380421f10bfff81d92583f5d05031bc166
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:46:34 2014 +0000
commit e7b5608b07627ca1f41df9ab84a7871894b959a5
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:30:24 2014 +0000
commit 91a44ad14e85d116362e7d70581e42b55c0206fd
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:26:24 2014 +0000
commit f2ea7651516f45ba675ddab28dcc165eb4af1eba
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Dec 17 01:26:02 2014 -0500
commit 857c945f3a4dea394951a6f4811dfbd570c915e3
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:02:25 2014 +0000
commit 77e0af9ba39a7574b22c16a95e79070cabb0778b
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Dec 17 01:02:14 2014 -0500
commit b43d2966d12d505840b34e023d77cc9da6cd8676
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Dec 17 01:02:03 2014 -0500
commit 7f967a4c9b4d44a1d387b8688a837ce0fc1425b4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:00:49 2014 +0000
commit 751a4939838ba670d41b3bd7580c83b2120cb4ac
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Dec 17 06:00:46 2014 +0000
commit 1e578f656e8e0f0af531dd9a4e040047a3466e79
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 22:03:51 2014 -0500
commit f10bb8d9728b7e8d65f649988e35afe6e65637d5
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 21:44:36 2014 -0500
commit 48bc7fa1e6cc4fabf84f1ddb6417e2709a154503
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 21:34:26 2014 -0500
commit 8ac6e6e6e1cc4c028b1a4aa083da4fb60831e7f0
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 21:34:02 2014 -0500
commit f1b866863b16157cba6c2449cf7e34be6b193cae
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 21:29:31 2014 -0500
commit 5a8839dfd02967fb2653a8a186f66907dd306f97
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 21:29:19 2014 -0500
commit 18d2a00f3e37a99ba723874449de808c737a58e7
```

```
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 20:54:02 2014 -0500
commit 57cb0c99f14a7886f7381d685bd1210b05f19c5c
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 20:23:34 2014 -0500
commit 5f37e0204afff5ad20670909d5421c5a055d5616
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 20:23:20 2014 -0500
commit fbbfb8762caf5e07301e6e8c20017cb447bdd772
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 18:56:21 2014 -0500
commit ac4f4329542853a298b434081bf64e1c8aa8c912
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 18:31:48 2014 -0500
commit 91cc102d2bf7bdc7a8af15c94a64bd15000f6eaf
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 18:30:42 2014 -0500
commit 65a950401b685f6e9a4b28edc60463c9de934600
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 18:30:12 2014 -0500
commit 581d63a1a547cac3c97b56e0b0904ca20d9e0b31
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 18:23:53 2014 -0500
commit 55949a8eae620aa754aa7adc35ea1a1c58999903
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 18:23:49 2014 -0500
commit 3ed0dea60d21f59b38b442803fd38b33681de4d4
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 18:07:39 2014 -0500
commit aec2c3e9fa5c1985024f8d925ce8a16472bbf86e
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 18:07:25 2014 -0500
commit 5b92e5d6261f16dc0fe0565f8656f7802f08d01e
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 17:56:21 2014 -0500
commit 7a6eda8ed33fb514fe29d9cced35580b61a50a99
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 17:45:55 2014 -0500
commit ea51b01d5a58806fdc613d863418364405c0fc08
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 17:45:13 2014 -0500
commit 102c6d689e26d1076a9b22174d1f6236119c6a56
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 17:45:09 2014 -0500
commit 41698430ca5b9980af090efcf8a41973678c8e15
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 17:38:48 2014 -0500
commit 695bce852a12093a2479527ff4026285ec1f7352
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 17:38:38 2014 -0500
commit 5b5ca08bab5a53696cb1a35dcb52736c72e6f88d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 17:19:10 2014 -0500
commit 66aa468fbd4a532b6e0e7edd80ecf35d1dfbc354
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 15:54:58 2014 -0500
commit 0aaf91cb41a66b79368da71289c1fedc8d969d88
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 15:54:44 2014 -0500
commit 2c7f45e7d39f9fdacd83c575538fb7d9a750f9da
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 15:39:09 2014 -0500
commit 46e475485865e5c8c61108fe3b3244dbb9035e4b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 15:38:14 2014 -0500
commit b3f04f703033e03e6218af3e8ee71f495f6924e4
Author: arb2196 <arb2196@columbia.edu>
```

```
Date:   Tue Dec 16 15:37:56 2014 -0500
commit 844fb12452709c3368703a952cc2ed237dde7d98
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 15:37:14 2014 -0500
commit af944360066cd4218e277f3f8e784e43f6f7450e
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 14:44:07 2014 -0500
commit a793b062bbe3c4574a5b30bbfd79da0212f66e23
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 14:43:56 2014 -0500
commit 3b6388b0f775bfa7921c3fc11c9c0483c77573cc
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 13:05:57 2014 -0500
commit 83e71f16ff962d9b62fc58ee64f91b0c5d7463df
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 12:59:44 2014 -0500
commit 6192b96c89f0b738eb9cec6c54d7a9230d418ca3
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 12:56:41 2014 -0500
commit 9e91a4601a7322a28e2b542b4ddf5a1a2b1b28a6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 12:55:33 2014 -0500
commit a8847eb6d7907e0cc525e4c5d03e6a404d5a10de
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 12:41:12 2014 -0500
commit 46b425e6e8144466178ad46b34943cea8e608f5a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 12:41:10 2014 -0500
commit 09caee45ae561662e17aa4a931743a5e78dbeabb
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 12:40:17 2014 -0500
commit f6e2529df0249a436d7dadad1bf91cb23021371a
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 12:40:02 2014 -0500
commit b728c76088c9ad2446749deff9b795c052a63d35
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 12:14:09 2014 -0500
commit c933ee03aba1203e29dbd38126d34424986e99db
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 06:28:15 2014 -0500
commit c457062bcc0c1bf2d001e39f52b87ddf98a8330a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 06:28:11 2014 -0500
commit e9a8785d62291cb30e5954c760d1606d87fa3b97
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 06:26:55 2014 -0500
commit 3658ead5e4431d7848c90cd0b3e7383b65313fb0
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 06:26:39 2014 -0500
commit a693e1b51072b4711b5ee5b947f4711160bf435c
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 06:03:34 2014 -0500
commit ec3b709a107e9d07d406e5318ebea501bc1ef249
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 05:32:04 2014 -0500
commit 8212553f32878bdfe5c4895fca64f20aeb816f45
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 05:32:00 2014 -0500
commit 69708aae83f179b3462d920d39c9d8648149be6c
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 04:57:10 2014 -0500
commit 3a08a272d1df1ab0c4079523c738250ca5841c98
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 04:51:48 2014 -0500
commit 85f441bcc79b634e85fb16fcc5ae01f632b7eab6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 04:23:46 2014 -0500
```

```
commit 73bbff0b213d675ee4a5241b4f6c598927d9fd1c
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 03:58:05 2014 -0500
commit 09cdc078efd819432790fb3109de0be3d799c642
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 03:57:59 2014 -0500
commit bcb96bab8376bba187c7ed7b53db8bdf2e9554dc
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 03:49:55 2014 -0500
commit 713bdc178fb0d2fd4b5d86ae7adf2fe99f6c9227
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 03:06:23 2014 -0500
commit e65cc767d9a106f75ff5853a7469979aee691cd5
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 03:05:46 2014 -0500
commit cf87fc8d74273ece9aafde9df143aac3cc56a0a0
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 03:04:50 2014 -0500
commit fdbd895d642bc26d5728d7a3e8a451ddaa90d614
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 03:04:43 2014 -0500
commit 5412116af8784fd330bd48a0b44eda734b75d57a
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 02:35:30 2014 -0500
commit eb50efaa2bad73d1d5c7eb67fdfd9a170248e109
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 02:23:25 2014 -0500
commit 3c2d26786afd0680b56216b62528686da3858138
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 02:12:46 2014 -0500
commit 0e1d4fc825c223882a6e1d8a4b9e162debbff4b6
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 02:12:18 2014 -0500
commit bc5ddf590a55192d7cbbe3eba6c17ff463bae89a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 02:11:28 2014 -0500
commit c068e1bcae7bc06642f09cb0150dacfd2cf57796
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 02:11:17 2014 -0500
commit e4e8c12c6da3d2ad21b5200986e8d1da30f12156
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 02:10:40 2014 -0500
commit 88455c11a92ada033d0fa5f138062cd696f46fa0
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 02:10:35 2014 -0500
commit d0b387cd942ab1ea4658f4f88dff5b16491760fa
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 02:10:18 2014 -0500
commit efefcf72f7e591e31c3dabc277a0c375deba26d8
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 02:05:24 2014 -0500
commit 409b56a81dcbbd6618afc106ca12dd053c702dc9
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 01:53:58 2014 -0500
commit de9aa32e8630b0c49984f48e701eae0d12ba7555
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 01:51:06 2014 -0500
commit aa57fbf9e1f41cf253c7019ab36df064f8ff06b1
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 01:50:46 2014 -0500
commit 3d1300f45521c2424422fb5d0999070f484a1d60
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 01:50:13 2014 -0500
commit 6351fba2b87245111f4854a7af3e2565403951b9
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 01:49:48 2014 -0500
commit 0c20a380f1e4afd2f760b9a0562f7983ac1c9e5e
```

```
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 01:49:45 2014 -0500
commit d29de47f9ed74ebe5cb3a43f18fdc124b7915ec9
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 01:49:24 2014 -0500
commit a7b8f3012a439d5a8a9146ea18a9214f79618d2c
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 01:41:34 2014 -0500
commit 826b33185254637f8994561fdfd9b6d3430079a6
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 01:33:43 2014 -0500
commit 76dce02d3db53beca674ccbf5fc4e93a24a1f91c
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 01:06:24 2014 -0500
commit 4d989798199938721f7d3cdc53ff2c6ce8cc1e3f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 01:01:50 2014 -0500
commit 26a836843d7a6157f5a80797b27b7ac86da9b760
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 01:01:41 2014 -0500
commit ae6fe11d283652e89823cacd34c548beac3ec608
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 00:58:04 2014 -0500
commit 40cc664b6df9f60890778b25edfadba71ca09d1a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 00:55:34 2014 -0500
commit c247735fcdf92263b30675dcb454bd75552eaf7e
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 00:35:58 2014 -0500
commit 9fc6338a1464ee0c221a6c8e073b05e4adafc592
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Tue Dec 16 00:35:50 2014 -0500
commit ffd3d0c1097a1d84656750d30255ed52eb979e1a
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 00:35:04 2014 -0500
commit 4ef786d9a20340e5e1478cffebd33cc8751654cc
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Tue Dec 16 00:34:59 2014 -0500
commit d1441055cca846635d3ca0dc6a64826b7d1b561d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 00:23:03 2014 -0500
commit d33ab12fdcf31c19ffae48135130945c48d2faa2
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 00:20:27 2014 -0500
commit 9daedc3c1473a7fe2db4be02d74754d9cc0b8978
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Dec 16 00:11:11 2014 -0500
commit 64e8ff9cf2a9f2999b587f5f1293589372731e5c
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Dec 16 00:07:43 2014 -0500
commit 49deea9ba25fcc87a2e50fecdf73c489885d3215
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 23:34:27 2014 -0500
commit d132c94088a398d5cedffbad92b6aa23129f2722
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Dec 15 23:33:28 2014 -0500
commit e5bb92c417a355ae2f9883e2bbf705c83858d8fe
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Dec 15 23:32:41 2014 -0500
commit 18e9aeb9a42824acdf2cba363d124fb02f0eb080
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Mon Dec 15 23:12:18 2014 -0500
commit 59a4a8b81dc6eb3c11459460c4f547d7ee8e606d
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 23:10:46 2014 -0500
commit 6290db206a401169a56813290ccc697e6bda810d
Author: arb2196 <arb2196@columbia.edu>
```

```
Date:   Mon Dec 15 23:04:27 2014 -0500
commit e2232d512103d59c938ed414fc44de675e55ed07
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Dec 15 23:03:41 2014 -0500
commit 655ac1f363a582739a1fa5046cfe7b7996002200
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Mon Dec 15 22:57:59 2014 -0500
commit 7de9d289a95ed5f64e73bb3d7db42abcd303ecff
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Mon Dec 15 22:47:34 2014 -0500
commit b0a58429ef6790261f1b480b415770df3a3a1f1d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 19:21:20 2014 -0500
commit 1b4b2777cd90776b346f31100bec83c08a568ab9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 19:21:04 2014 -0500
commit bedce228c21bedfe740136b5d784fbb867386385
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 19:19:49 2014 -0500
commit 61f2cfcf362192a3d3a8ced5a2eaca677f1ae963
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 19:19:17 2014 -0500
commit b501e584c594bb5d7a2ba1c5c82c39f83f8da842
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Dec 15 19:01:36 2014 -0500
commit 89eac206e3a35b4c0b79d96e92b5e60e0d3f2a5e
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 18:59:41 2014 -0500
commit 56382d8596b9232c2e7e27f788bf031ad6aebe98
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 18:54:48 2014 -0500
commit bdc08c458e3c1d681fe0872a3ad0e8cd59850571
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 18:54:18 2014 -0500
commit 0256c3adfe6ff92c19b9fab3bf99ba88532a8bac
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 18:27:49 2014 -0500
commit a040e44936afa3a2e21db23486061401ffb82178
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 18:21:50 2014 -0500
commit 2e56911f55f3bd59c33a214133c8260781e52f54
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 18:12:31 2014 -0500
commit 7e50110b791104db854ddc8ecc1a24630b03e963
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 18:07:22 2014 -0500
commit 4081005cd02a4578176a79c645d992b52f7ccbc2
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 17:53:04 2014 -0500
commit 52e3da0d4f553766f4aa5563446dc82092946045
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 17:48:12 2014 -0500
commit 9dfbae5e5307098b3fa6acc18f2383b80b49ac73
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 17:42:50 2014 -0500
commit 9eb649a9122870bb9a665b373d971438d4818b93
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 17:23:30 2014 -0500
    commiting a broken gen
commit ed9679a5ba30aafb2aa599b7c63bb77c267b5996
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 17:22:20 2014 -0500
commit 566b31f2f2739879654e3ed662c528420d1765f3
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 17:17:29 2014 -0500
commit 0700cafc45d6198d2a4a82e178a4f5c99d951c68
Author: Benjamin Barg <benbakerbarg@gmail.com>
```

```
Date:   Mon Dec 15 17:17:27 2014 -0500
commit e79f07175c2dd77a595962a2432f1980f4dabb83
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 17:10:35 2014 -0500
commit f7089e6bb2c808ae433927f6cd2d37408eb952d1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 16:12:50 2014 -0500
commit 493c1693e2dda5195bc1ac886e364adbec736e89
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 16:12:23 2014 -0500
commit ab34eb898f72d12e004e898f1dc1250a57d6d9a2
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 15:54:47 2014 -0500
commit 211a4ff81e5064b19c75b9e6f293b6c63b6a80e8
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 15:54:44 2014 -0500
commit 2129ade26c499b5d3a791bebed17d50bde301baa
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:54:04 2014 -0500
commit 82270e3dc2a54cd2a0e963cfaacea7de525d3802
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:49:08 2014 -0500
commit 60fb9599eaca9eee820723ee9894c7011fabd772
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 15:37:25 2014 -0500
commit b0e4c8e042e7c870bd173ef2023e8e5a6ef5953c
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:35:53 2014 -0500
commit d2b21866c3f1d8945ebf12ed06fd71be8f166a1a
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:21:54 2014 -0500
commit f3e0f767f05e863ae69b9525b876bc053388c410
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:10:58 2014 -0500
commit 3faf34ed41a1082f2dd017f07e333395d39b8b9a
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 15:05:49 2014 -0500
commit 71cca1a70ee5179809cfbd81f8e1a3b58cdec01b
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 14:18:08 2014 -0500
commit 7bfdf84b402a88672c0bea84f16856ff37843bdf
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 14:07:07 2014 -0500
commit b5e97db20f0a8cac44bfa8e35b97d500cf35e94b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 13:34:54 2014 -0500
commit 6cb84ee81080c21ea0d1941b449359c60df13641
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 13:34:49 2014 -0500
commit 0e25e1eeb8a1fb2674ea0aab3ed0ecaef661ad26
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 13:33:47 2014 -0500
commit 783083dc0785149d60d001bdd995c8292aca2efa
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 12:41:53 2014 -0500
commit 49f42aa1d9960ef7914c20ca36831b6e90a029f7
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 12:29:43 2014 -0500
commit 287985e4ccf676e3185968e67d2255f12dd50f68
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 04:07:54 2014 -0500
commit 9178c479b2d8246e218231742a494e59e60e068f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 03:32:02 2014 -0500
commit 74048111d3bbda197c5bd991491c9ab2fc163856
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 03:19:56 2014 -0500
```

```
commit aa3e3f33e97b96b79fef59bdf2b6110b852af3dd
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 03:10:55 2014 -0500
commit 383fd7fe2bb33dc08cabd7638daf1e2b525300b6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 02:54:53 2014 -0500
commit 686e0b53a9104d2fbf55d47c28ca12883f26dc66
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 02:52:13 2014 -0500
commit 050c1bd99346074c63c60fb34b342765ed7db785
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 02:45:39 2014 -0500
commit c12afd6609fc76edfef24a6e73de6031bed90ad1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 02:45:14 2014 -0500
commit 13c761cd26a5d5620f5f5e44e2f0621b85aa5b48
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 02:43:35 2014 -0500
commit ace9f3d1f8f16ccdc37f396f4adc36eced4ae058
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 02:41:40 2014 -0500
commit 60b75848f188cdb879f1acf4b73f1b73e7389182
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 01:06:28 2014 -0500
commit 1fa84473b0b919bee020a29e62aa4608f6396b40
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Mon Dec 15 01:06:26 2014 -0500
commit d9314c843bcdbe894ad6318d77131a4e127b89bc
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 01:05:51 2014 -0500
commit 4cc5756a313ef0225ef5dc6078ee706d2426a438
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 00:36:45 2014 -0500
commit bc3d30251f478f57b08e3a8706401ef2ddbc14cf
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 00:30:15 2014 -0500
commit 65902025ee11b0ca777ef05980e968a09e91622d
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Dec 15 00:29:54 2014 -0500
commit d0a16244eeb70b217b138cdedceb33e2aa976a2f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 22:42:48 2014 -0500
commit 019d25804b9730dbf606a74628e377a58a8204ff
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 20:48:57 2014 -0500
commit 3f638357aabaf0775ee20afb1790b330a66c184b
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 20:04:26 2014 -0500
commit e478cb2bc3f3e093ef05230d99ba74d933892541
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 18:51:33 2014 -0500
commit 1f294b7f44bac2e53848b5c11789d977d90f283e
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 18:51:01 2014 -0500
    cant commit wo breaking the build sorry
commit 22cf606f41eeb3473989cbd8a2753206a609185b
Author: arb2196 <arb2196@columbia.edu>
Date:   Sun Dec 14 18:24:52 2014 -0500
commit 01613fc2e3a992176d16e33dee4045918c5d1cd2
Author: arb2196 <arb2196@columbia.edu>
Date:   Sun Dec 14 18:24:10 2014 -0500
commit a59e8bb63749bb767d30c7a1b4121322c737747f
Author: arb2196 <arb2196@columbia.edu>
Date:   Sun Dec 14 18:23:42 2014 -0500
commit a594fb7405f478bd8a4a7b1e2704a45e9731dcf1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 17:44:53 2014 -0500
```

```
commit 4975acb5317768a08fb21bbae65ec8821c442ae9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 17:44:50 2014 -0500
commit f211477e3e29d91e6b64d0d8723c366eed879541
Author: arb2196 <arb2196@columbia.edu>
Date:   Sun Dec 14 17:38:56 2014 -0500
commit 20fdde985077b8c39b13dc8633a60a39da477ebc
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 16:57:43 2014 -0500
commit ff80b96794007ea642d7451eca396c2a80140b2a
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 16:53:26 2014 -0500
commit 2cc9b52be5d76918ff9629c1bbee53679baec3dd
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 15:21:27 2014 -0500
commit 3070b21bed5c9fd7f10c4e421479c3f244ee1992
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 14:06:06 2014 -0500
commit eca3384928c1de9e04644aca4a1f50b291ef071f
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 14:05:50 2014 -0500
commit 4ad1c7e700e5e116c2b71aadf6ecde1ee8cc5430
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 19:03:31 2014 +0000
commit cef621a968d5d343aa0dbbfaf3d47923b0e6ae39
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 13:23:36 2014 -0500
commit 4294a65a7487170dfc11139a0f008abf0744331b
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 13:17:17 2014 -0500
commit 4c6840881c0000a7d1c57af2382726f05320d89f
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 13:10:05 2014 -0500
commit dad579ad2e4da814ba7b1f098345c86c5ae78c5a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 17:48:27 2014 +0000
commit 7ccfa6e63ab5c7494d430da8e26cbb3b166df340
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 12:32:09 2014 -0500
commit 143855e8913cdd06aba1fdc9cb63b702162f5923
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 17:19:51 2014 +0000
commit 8bed87a21831b00f487d1a9df6098047809d7c18
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 17:19:44 2014 +0000
commit d18a4b13a8b0c93f12ec7b6beefed9a15316d230
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 12:18:16 2014 -0500
commit f304f391c95318086bd49da13b429160bee7a5cd
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 12:16:38 2014 -0500
commit 836166c4afaf5906f00cc4b7cda0bed382b462c4
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 12:11:04 2014 -0500
commit 2b1949ab51743124e7a223ef6180481b446badb1
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 12:05:01 2014 -0500
commit cd2b912b0a0f04a53eb1a9adeacf1e2fa44026d8
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 12:02:13 2014 -0500
commit 236bc6b91b49ffefb75fd010ac2423828cf5d404
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 16:49:53 2014 +0000
commit bba97aba407c61b1754b5476840e9fa5f25aa9ec
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 11:42:38 2014 -0500
commit 1e43011f157e284f40badc822f83b8831b0e75ec
```

```
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 16:33:27 2014 +0000
commit 817077932a8774f97d42bad4d9c9a2b98b10cb55
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 16:08:54 2014 +0000
commit 9816371464f950b6dcc4acf14163621e5f54f74b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 16:07:33 2014 +0000
commit b186c64975ffefb0aaf0cf2f93fa01a6f6837e82
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 10:37:13 2014 -0500
commit d92dbb5aaaf37425c560d639926058b1778f28c8
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 07:53:04 2014 +0000
commit 4b13823c8b7e34a4d026b348dbf266cf649339a9
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:51:58 2014 -0500
commit e13dba939697b9c3e1c075c2f8f044fa5512f8f4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 02:28:42 2014 -0500
commit 05e99b5d022a5d4c958f20817b64c110c92e7ab8
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 02:28:24 2014 -0500
commit 82f50a233df5720115ca5cf0941294cb79ca0a1b
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:27:26 2014 -0500
commit da2c1cd41497dfa3a36c12f014a57acde539991c
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 02:27:08 2014 -0500
commit 106aa9305c70bca09eef1860c5f83d9e28ad7a99
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 02:27:00 2014 -0500
commit 70534850c1eef5a2715e6cdfc2dfa1a3cede0f4e
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:20:07 2014 -0500
commit b364ed8af3cd444aeca96ff280611c9381a34ba5
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:19:26 2014 -0500
commit bcc5103ccdfa148548328d6a89729d07b18f2e23
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 02:10:58 2014 -0500
commit b03d40840bfe2d1ea046d92257ca67031a3d9b76
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 02:06:20 2014 -0500
commit 5d31c063079664a2eb52d0849d5d690e8966148b
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:02:57 2014 -0500
commit 9ff36fbfab0d535c83ab9e4224729bc4ab29e274
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 02:01:34 2014 -0500
commit 748c5e3c9fece47838f733871761dcbe9242fc11
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 06:52:03 2014 +0000
commit 676367aa73b7ab50043527b32e8d1f880be2691c
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sun Dec 14 06:49:21 2014 +0000
commit c0ae34f51383162ff1d6ed31c04b6959c8a8d447
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 01:48:36 2014 -0500
commit 9512929a23e4938f7cf7651aa90f3baed4a6d835
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sun Dec 14 01:45:18 2014 -0500
commit 7b89ad325ac1f12b4782f8f3260a10ef6543e1cc
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sun Dec 14 00:32:29 2014 -0500
commit e4202e36ff2c9197d20f47a8475d7e7d0e7b227d
Author: EC2 Default User <ec2-user@ip-172-31-13-130.us-west-2.compute.internal>
```

```
Date:   Sun Dec 14 04:04:56 2014 +0000
    removed accidental bckups from commit
commit 3701388a6252996362cab180304f0dbbf507fb5b
Author: EC2 Default User <ec2-user@ip-172-31-13-130.us-west-2.compute.internal>
Date:   Sun Dec 14 04:04:33 2014 +0000
commit 37d91f7e58a387b1e300e855ee9a9e4df1fadd7a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 22:17:13 2014 -0500
commit 20cf20d8c71ad30c558e301c9b563b2f1ed26b66
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 18:54:19 2014 -0500
commit ea4209f3064999de6685760878c17bf44d39004e
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 18:43:24 2014 -0500
commit 0c25770368b3a0502f0629562acc4367706eb0b1
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 18:41:36 2014 -0500
commit 637fbd488196e3ffdd5ffdea64a272ffa1a3c635
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 18:15:43 2014 -0500
commit 425a1a106e1b4c63ab2a00c589172e585f511703
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 18:07:28 2014 -0500
commit 72fb1bccf2952e6450b76c921df83210bcc397de
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 18:00:40 2014 -0500
commit 78369d6b41d9e8696aad30cee305cabb9ec5a5a6
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 17:58:36 2014 -0500
commit a547aa7e46617c519c111709adfcdf76406a8886
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 17:57:53 2014 -0500
commit 71dafab1453f54b3028e9a072717e9504bf0d26b
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 17:47:37 2014 -0500
commit f00406c1fa392bc8a64a026448595c07bcb81c9c
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 17:43:29 2014 -0500
commit 639fbf2793c7de6afefd6989be44d0e5315a35d8
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 17:42:33 2014 -0500
commit af154a216642a0baf6ec473f6eae5a268164a621
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 17:36:11 2014 -0500
commit 5562c4b0bf508e0f2b8d901a4c7d365e3842a970
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 17:22:07 2014 -0500
commit 90303fd948a720706260053600 0c9934d14df663
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 17:08:08 2014 -0500
commit 992bcfc40e51dceb41f1f66ff29fc578e1b8b549
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 17:03:39 2014 -0500
commit 0695080d35d64f95316d6a0cf21dbb8ce9164d23
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 17:02:40 2014 -0500
commit 4cccadf0e73b0b1cb5b6a0f9174e2cf1d6a8b867
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 16:55:32 2014 -0500
commit 78c36ae2bce0c6429cf8c8a5f765f01665e34f98
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 16:53:45 2014 -0500
commit 86def36cd8d515fa21fe682c31a74e144d7fe512
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 16:52:30 2014 -0500
commit a57af13813dfa4fe3538c70339a690d1432d30e6
Author: Benjamin Barg <benbakerbarg@gmail.com>
```

```
Date:   Sat Dec 13 16:52:17 2014 -0500
commit f72cd5e73ff1d013398cf791749868355b7ecdc2
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 16:50:21 2014 -0500
commit 7f387f767080111074ec9e4f6b8cb11bf8b4f229
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 16:28:23 2014 -0500
commit 59508f5970ef899d22d34bafacab874c30ac194d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 16:15:38 2014 -0500
commit fb91331a6214fedbafe9b5b10819ae85fad8ea28
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 16:15:22 2014 -0500
commit 617850fdcc01c5772c071e8adc1f78ef25d9ebd0
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 15:25:26 2014 -0500
commit 04b92e1931bcd0d91717154d5d0b7ae8f5a4aa53
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 15:25:17 2014 -0500
commit 8e21cf2be21e02cb98e39f278454f010a21ea347
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 15:21:29 2014 -0500
commit e616b72c81cc350e7b04590e17c4d7644ca1474d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 15:19:50 2014 -0500
commit 6b05b77aa461ddd419d93f42f32f35c8bc51cd4c
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 15:19:39 2014 -0500
commit 85346223f10336de35f1dffae033ad75f01585fb
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 15:09:59 2014 -0500
commit 123cc86ae17df020d384499900d0924aa4cd32c9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 15:09:53 2014 -0500
commit 07fe49a216184f2b22e6b68e7e570ae543075781
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 15:09:32 2014 -0500
commit d51328e7de56f9fb013c31828348c6739a1a2fcb
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 15:03:43 2014 -0500
commit 7be0b79e52eb27ce49512030711d9296072896e5
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 15:01:50 2014 -0500
commit ad58daba97777491bc6ecfd54cbcafb756749c4b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 14:50:05 2014 -0500
commit 3a562c930d033a48cd1017cda471ab674d0b1afb
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 14:49:37 2014 -0500
commit 131c564bdc36b81bc98e48e47be0c488567391f7
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 14:49:17 2014 -0500
commit d87892c4d99288b42c16d581febe478f20a474e8
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 14:38:14 2014 -0500
commit 97d82bd76d687b35b680fff83d716c810b66dec1
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 14:31:15 2014 -0500
commit 8f0c378b545cd9845d579a89d86fe92e070ce9aa
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 14:27:22 2014 -0500
commit 01a4a9f36e682794f1eb7d0c120bf6cefe61f7b6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 14:23:06 2014 -0500
commit 19fb9d9dd395554465696147504e1d35534e0ae1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 14:22:57 2014 -0500
```

```
commit 1a70c6ada1689f7395816c74945e7d64b8942b37
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 14:21:44 2014 -0500
commit 25a7637790e5ea2cb921cd8b9de282b1ef73cc31
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 14:21:10 2014 -0500
commit 134cd881ac5a197b6186ee3b0d13c326b81a568f
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 14:17:42 2014 -0500
commit c82b60582114d97ca97aef1678296d60f3750fad
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:58:11 2014 -0500
commit ad5074cc2f2ea53f3327aab3e9b8fd36e2c595e4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:57:14 2014 -0500
commit 16b094ded7684e813d7a6ed64535fb8763e2887d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:56:25 2014 -0500
commit 53d57e0b9abe3654fd3b27ff08f654c631f608a3
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 13:48:13 2014 -0500
commit 0be398ce85456ddabd04c4877cfdbe52e28634ad
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 13:47:52 2014 -0500
commit dbddf08df95685d76d187c9ce89cd87c4f757840
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:38:35 2014 -0500
commit acd8c3c2ceeb252bd4ff0939e0679c31101d61ad
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 13:37:22 2014 -0500
commit fa72c5ca193f9d6726dd7ae49b83514fb863146b
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 13:32:15 2014 -0500
commit ec791cfd134e8a3244aa78021d6b6ec2816ea4af
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 13:24:24 2014 -0500
commit e43bda2e66f95c8b7cbdc995f4fe6d4fee8bad49
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:14:59 2014 -0500
commit 1c314e70a5aad353615945b9c98e7711c15ab0dd
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:14:15 2014 -0500
commit b13553153a7b1d3c1691d5bcff0cc3dfb375e8f7
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 13:14:12 2014 -0500
commit 94c43d644194e012ba45aa3d1c8b1977d82119fa
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Dec 13 13:13:24 2014 -0500
commit 0c0bb37466b68cbb6b74cef830ebbfb91943cbd3
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 12:57:29 2014 -0500
commit 3c165de1f6096f3110733cb9a62c54a13e04c3cc
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 12:57:08 2014 -0500
commit aeeaf3aed920af97a1dfe2d0de52b6979ed24e86
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 12:56:55 2014 -0500
commit 5cbe5fd0de6a751db2e722a2252b5ae95910e044
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Sat Dec 13 12:23:20 2014 -0500
commit 5e4cb643f62aefaae3ac5de04415dd7728dd2fa4
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 12:03:21 2014 -0500
commit 68d9431951efc11ff7553e58c004def4e927fedd
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 11:55:32 2014 -0500
commit a20265e1842b19997052028e78282c2442389247
```

```
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 11:42:53 2014 -0500
commit afd43705c20da8eca0ea7e1082b5d1afdd65ed81
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 11:39:38 2014 -0500
commit db80bfda6c26c734c9a9207246f6741d646a2c10
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 11:27:35 2014 -0500
commit d0c6292a16083bdb3ec3b9a1f8b53f94095a667b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 11:27:12 2014 -0500
commit dfe6a4b7fa4228128b2dab3cfc3d7a34f9465be7
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Sat Dec 13 10:39:06 2014 -0500
commit 4696a82a603cba7e6dd0006992eeafb081c03d93
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 10:37:53 2014 -0500
commit ef3a9acbb41df949ce727c4360b6c7d9550db9db
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 02:44:11 2014 -0500
commit 06a3561e683724a0924332fe33ae2692444e549f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 01:58:53 2014 -0500
commit 7e98f71309b166ff3b4108e09d2495976109efe5
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Dec 13 01:37:56 2014 -0500
commit 016f95e96032268711c62d93cd9303c5c0f26cf8
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 23:45:14 2014 -0500
commit 1cf03d73043ac4d4300b87fec5bc7bc2dd2ab67e
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 23:44:12 2014 -0500
commit 0d9a90ad15bf4e5242f0f33636dfd01653a26292
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 23:43:38 2014 -0500
commit 9baa7fdf0857577b2acf468f45fa9cd0d81b95d3
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 22:05:38 2014 -0500
commit 60534ce57d944336619f61c09447e6fccb849908
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 21:57:39 2014 -0500
commit d2176d83dca5079043de76b6d3ab8a85330c680e
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Fri Dec 12 21:21:45 2014 -0500
commit 33ef2ece3064d33381df0b6e25c922c335000d83
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 21:15:21 2014 -0500
commit 10dd97ede739ca0d5a4ce9f695005c988a15f8f3
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 21:14:27 2014 -0500
commit 6900df650a10c42dc5e3f9b42a6954ac25dd32a0
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 21:12:09 2014 -0500
commit 435629451cb0405d980af0da63775602734f6f7a
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 21:10:57 2014 -0500
commit b622b326b7dbf584621753d343a630d1e93a4105
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 20:32:22 2014 -0500
commit 35aa5aff70e6c17faff70739535c0129d0d1ff3a
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 20:07:07 2014 -0500
commit 8e0f124487f1b46bd58470894653e3e2359b1a3d
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 19:28:34 2014 -0500
commit 589c23c69caaee93d7494db9ace2b79822e3a580
Author: Gabriel Blanco <blancgab@gmail.com>
```

```
Date:   Fri Dec 12 19:28:01 2014 -0500
commit 0519cc477c21433c6a45ddd610de9f8d2e3b3aff
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 19:27:09 2014 -0500
commit 214ca04c050a724918d60fdb8f24cda2941803b0
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 19:18:52 2014 -0500
commit a99de940d0e607002588141900941939ef1df799
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 19:08:17 2014 -0500
commit 4e66c0e455a640799944e348a8afbe51211717e8
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 19:07:32 2014 -0500
commit 6a04567ec08076205fec8565cc03e9b9ab22e5ab
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 19:06:21 2014 -0500
commit ae2717b99ea0bba23e01cc244bfc19358105b333
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 18:47:33 2014 -0500
commit 39cee3cebae8aad9f99182c71d5bc3009fcc937a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 18:33:09 2014 -0500
commit 412ca209b0b0a3706581ab9c7df780eaf1fccacf
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 18:33:06 2014 -0500
commit 459982404f2615b6966372f3274848e1beac7840
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 18:32:10 2014 -0500
commit dfa57f765b73db8c359642c50308cf740a378ac6
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 18:31:05 2014 -0500
commit 02e114f823c8aefc923165843a775f416345c238
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 17:54:33 2014 -0500
commit a6c4239696066a517caf42eeb03e537823c9f127
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 17:54:28 2014 -0500
commit 690770bb4e22f2b1ad01cf74282f765bb1d8efd3
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Fri Dec 12 17:46:18 2014 -0500
commit 553b0473d02af0b8d8fca888f8247b3ff87da7e4
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Fri Dec 12 17:43:51 2014 -0500
commit 075851084844bc337971e99e9a25b2c15230af28
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 17:42:11 2014 -0500
commit 9db656e9466670ec604a20009da4503d6c25f292
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 17:23:45 2014 -0500
commit 4de9a2dd556f28d6d4d6c45f508e6c599f4fb9c1
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Fri Dec 12 17:20:56 2014 -0500
commit a6d50251a02053c838000121fcb429a6408154b9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 17:16:34 2014 -0500
commit 7e18b4c9b2bf15525632ebd53f610be20900b099
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 17:08:40 2014 -0500
commit 7fd40ff993b5b702e22f65cb25cf197342d07570
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 17:01:14 2014 -0500
commit 3b176704b5a4d5a1e0e8e7d982e8f7addbdf1a16
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 16:10:52 2014 -0500
commit 99bc3d7e178a892b4c84c7b7512c073d4aae5ffa
Author: arb2196 <arb2196@columbia.edu>
Date:   Fri Dec 12 15:48:38 2014 -0500
```

```
commit b5b70e1e7ca993bdbdc8025fa09449e4c5460541
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 15:46:05 2014 -0500
commit 7d6f74c3baf5eb3179e129b129357779fe87eda2
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Fri Dec 12 15:24:44 2014 -0500
commit ebd1acda8e53c241bc91594884d45785c6eff725
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Fri Dec 12 15:21:55 2014 -0500
commit 637d5083bd3c26c67c54d89bd5b955814049cb28
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Fri Dec 12 15:21:25 2014 -0500
commit 96dce5c3c291e01a9818195e018804fa3a85424f
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Thu Dec 11 22:16:38 2014 -0500
commit f4d9d1f2b75b54cc3953ae42fc95abc5d09f33e5
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Thu Dec 11 22:16:22 2014 -0500
commit dd1e0041b3158f494b31a817a5bbfcf5474ea9fc
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Thu Dec 11 18:40:04 2014 -0500
commit 1c3af77d4b8d4e4ab6d4648a68f8a89c05213e3f
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Dec 11 07:58:03 2014 -0500
commit 9aca9007e3c7eb094f362053ded812f6e0fa623d
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Dec 11 07:57:56 2014 -0500
commit d4383bfad7d3716b37882c7cf95b7b9f995b6451
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Dec 11 03:25:47 2014 -0500
commit 1bc310cacd80bc1e2f5902eef468f89f7ba9eeda
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Dec 11 03:24:16 2014 -0500
commit 2a6eefe869a161c6d771c06e0a5e7dea16d833ad
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Wed Dec 10 19:06:03 2014 -0500
commit a88b0dac419ddf31d64657844f23b2ed2fc231c0
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Wed Dec 10 18:44:16 2014 -0500
commit 3ed37a0406664b8c0d3f7b12142eefd088510942
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Wed Dec 10 18:43:03 2014 -0500
commit c24305d25f939ea4216cca68e4de9e4623eee02f
Author: Gabriel Blanco <blancgab@gmail.com>
Date:   Wed Dec 10 18:41:00 2014 -0500
commit 8d41239dce4a9cc2fc7fcda58206b4e881026054
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 10 14:54:41 2014 -0500
commit a382cce3949ea452e494fac9b0c0a7a6d6647b7c
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 10 14:53:54 2014 -0500
commit 6b4ea31571da66e6993fefafd3f88b07d3f57c84
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 10 14:53:21 2014 -0500
commit 7ab7db15a305bf7abb963430689c2b5adbd4a73f
Author: blancgab <gab2135@columbia.edu>
Date:   Fri Dec 5 11:42:23 2014 -0500
commit 3004927c777be380b86b5e93774afc110c039656
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 17:48:29 2014 -0500
commit b21fcee582fd1b5df88191514104029da06a5a31
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 17:48:11 2014 -0500
commit b62299b7bb4be5af97f76690328b9ef3e828b581
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 17:15:14 2014 -0500
commit ad20868c05beb098b40a4b740ad99f9ef108e05b
```

```
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 17:04:46 2014 -0500
commit bd467eac3e60c8c496db08a31d4c563806ee51b9
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 17:03:41 2014 -0500
commit 0abfdf8698f535bbd97d98440d0e338f94e086f4
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 16:59:49 2014 -0500
commit 05167095fca028b906e7b12cac4083ea0fad7669
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Dec 4 14:56:30 2014 -0500
commit 83aa4766f60f425405c2bb9589bc953abd3d6e94
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Dec 3 21:39:56 2014 -0500
commit 79494b139e8abe8d8a946449ffcba38c264eaa47
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 3 18:10:29 2014 -0500
commit a402dc9ca4e565101daa43ce67620e78c0b82e68
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 3 17:52:10 2014 -0500
commit c9a9fb4441af34637d859bd54b36d8dcf1a341ee
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 3 17:51:35 2014 -0500
commit ec8930dfc0f8ed90df7377df5909eb9d3264df2d
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 3 17:51:00 2014 -0500
commit 69a60d8e5f3faa232dd71712a2c9abf79b5c7d53
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Dec 3 17:50:38 2014 -0500
commit 7643ca836437b33ac779db01533fc39ec443e6eb
Author: arb2196 <arb2196@columbia.edu>
Date:   Tue Nov 25 16:43:56 2014 -0500
commit f5b7adef389cf91d48fce84cf828b6576d1d1e27
Author: arb2196 <arb2196@columbia.edu>
Date:   Sat Nov 22 22:54:05 2014 -0500
commit ce1161c294cf9cf925ebb06a1dedd6c1c0e04a71
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Nov 20 16:49:04 2014 -0500
commit 3fe5a1ba6823af20d49e49b84cd59cf833fed0c0
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Nov 12 18:35:05 2014 -0500
commit 7553c1cc2beb709612574d5ef978173c7429dbfb
Author: blancgab <gab2135@columbia.edu>
Date:   Wed Nov 12 17:25:12 2014 -0500
commit ec418b043ad0f3e1a2e1bf6e3f1c006637791be1
Author: Benjamin Barg <bbarg@users.noreply.github.com>
Date:   Wed Nov 12 00:06:27 2014 -0500
commit fcdea6e10544c8271e28155cbd04aa3716cff347
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Nov 12 00:01:49 2014 -0500
commit 3f992e810a3c12b33206dbb6b3884796598b3c41
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Nov 11 21:05:01 2014 -0500
commit 89c8a1a6754cd9c834c4eb2ee41f6caf31383259
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Nov 11 00:20:57 2014 -0500
commit 84c07ba7fa937cfb1cde19ee2652b1379e388900
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Nov 11 00:20:21 2014 -0500
commit 69dee01671437cfb7d485895d97c4eb0177dfd11
Author: blancgab <gab2135@columbia.edu>
Date:   Sun Nov 9 11:01:56 2014 -0500
commit 1ee1219e941d3306b4c37cf7d0ccefdf85aa37c3
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Nov 8 18:59:53 2014 -0500
commit 1600fa9763e5d87fcfa78dbef0a7a9162a209c2a
Author: blancgab <gab2135@columbia.edu>
```

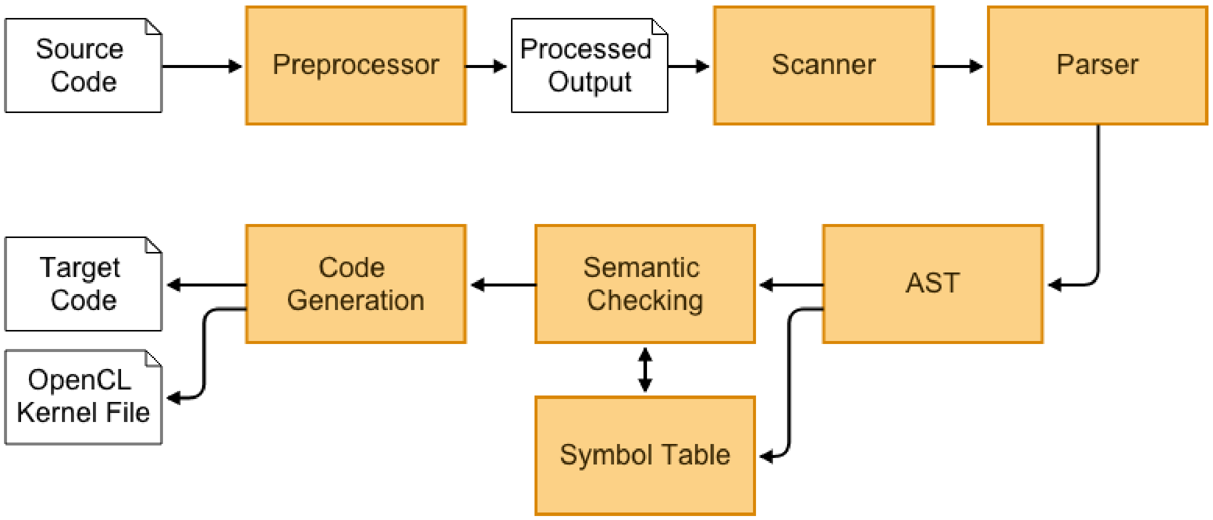
```
Date:   Sat Nov 8 17:39:42 2014 -0500
commit 10e68c7197846add8d632490631d0bf73b4ee979
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Nov 8 17:27:35 2014 -0500
commit a565383d4a48107855fb44961cf07d437805a94a
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Nov 8 17:23:38 2014 -0500
commit 168f1a0d449bf256119aa50cedb7b8383a1de534
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Nov 8 17:16:01 2014 -0500
commit 1c384b193f92b296dbccf6e654a8d362af15b001
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Sat Nov 8 17:00:17 2014 -0500
commit 2b50e766c89d5b285a69de4d9fa9800fbd4f8b09
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 22:21:33 2014 -0500
commit 44748efd5084efe2870712f7f05784644b049f5a
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 22:20:50 2014 -0500
commit c00248b539ebb368e0828ce83e26fa7dadaff857
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Nov 6 20:01:57 2014 -0500
commit 2b1ce539bb61660ad2312b99ee8f11b114208adf
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Nov 6 20:00:51 2014 -0500
commit e05082daae379068ee7ff948858c19addf6c60d7
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Nov 6 19:56:11 2014 -0500
commit da64b2ca00875a29ce9e224c1b075a826d010869
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Nov 6 19:45:33 2014 -0500
commit 5cf77ab1d2224aec17026bbc1e362eee5c6f2936
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Nov 6 19:44:48 2014 -0500
commit b5bdf8697df1baf73a7f536c77f5601c92ea4ae6
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 19:34:35 2014 -0500
commit 4e3814a7a8ab656c27afa7d81f1c03c02c6beac3
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 19:00:04 2014 -0500
commit 0cae61f4bd97866450e5851f2e2e5a10ce2a2bda
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 18:48:22 2014 -0500
commit cc33da3529efa7ec8af221d7eee134b278d21cc9
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Nov 6 18:46:40 2014 -0500
commit ae0f7c39e9ae66ed32e5b9ad89ea2c3fa8246872
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Nov 6 18:16:02 2014 -0500
commit 43d63ddf5ed1e97bed50782dd6fc6b060e8237ee
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Nov 6 18:15:56 2014 -0500
commit 7466071804ee3bd3b40c99d8e32bf166d98c9876
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Nov 6 18:15:08 2014 -0500
commit 9fef7d760ed11337e18a5557f2438aa973fde693
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 17:56:39 2014 -0500
commit 141e926bd5a083d907c0ded22df60b2ac9b9e121
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Thu Nov 6 17:42:42 2014 -0500
commit 356791d5def1d8cf8adb09c7977a51ef4b916830
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Thu Nov 6 17:42:01 2014 -0500
commit 016afd73b1a08ac2aea107f28e636b9a59ae941a
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 17:41:05 2014 -0500
```

```
commit 6b1b64c6dd7bb22c4c42fecd70206f84396d1e1c
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Thu Nov 6 17:39:22 2014 -0500
commit 722ac775d6468bbb43849f96668689cb60cddd41
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Nov 6 17:23:02 2014 -0500
commit 65371383ca1ca265a8467d46e1bdcafcc7fd27ed
Author: blancgab <gab2135@columbia.edu>
Date:   Tue Nov 4 13:33:45 2014 -0500
commit 5051ae040dd099502f49b2fef8893d4e45984bbe
Author: blancgab <gab2135@columbia.edu>
Date:   Mon Nov 3 18:20:49 2014 -0500
commit 88eba2dfdfd4886b4e02f1cb16ef24a2b996e59d
Author: blancgab <gab2135@columbia.edu>
Date:   Mon Nov 3 17:38:07 2014 -0500
commit 2bc982bb5ca937e26c5d4240d91b375b9bb48782
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Oct 27 23:34:39 2014 -0400
commit 118f6e71e7a8e77bc0a31e678c8ccf59a02a35b5
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 23:30:44 2014 -0400
commit 7fc2066018a97d712f07613cde515c7f487042e2
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 23:30:28 2014 -0400
commit 20f86f1dac1edf5c28970c08664a945e401610ef
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Oct 27 23:28:13 2014 -0400
commit 2b6277f198c18fbeb6beaff7be3d199605d1b7d8
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 23:21:17 2014 -0400
commit 58a6ecae01db8aea2811f8f0dcd6821787d5399e
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 23:21:07 2014 -0400
commit 46d2d47920e7c360b23603c6c8988e21f1bcccf0
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Oct 27 23:20:20 2014 -0400
commit 0e69cc3e898282a785ce37853d12267ea1608b41
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 22:54:46 2014 -0400
commit 091ff505a3b6c5f7b1554256e45b016ca03c12e6
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 22:51:17 2014 -0400
commit 4f2dc767381909b3def72b9b0bff597c0cb7dba2
Author: Ruchir Khaitan <rk2660@columbia.edu>
Date:   Mon Oct 27 22:50:49 2014 -0400
commit 05e7230bb9d8b000af83561adc909b5625adde90
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Oct 27 22:49:51 2014 -0400
commit d0f851cbc504e2350494835b54f95a9839a8f229
Author: arb2196 <arb2196@columbia.edu>
Date:   Mon Oct 27 22:31:49 2014 -0400
commit 198d5a0729e0ee9a9985e8c80a7e94208aa96e01
Author: blancgab <gab2135@columbia.edu>
Date:   Sun Oct 26 00:13:51 2014 -0400
commit 95a5ae793dad79d79027fe3f2c79dbf62b800cb1
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Thu Oct 23 19:50:11 2014 -0400
commit ad397b5d72f4f41785113878d28b0e2da5d3ec09
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Oct 23 12:38:44 2014 -0400
commit 6ffff929e83c70432cc7a585e075002689e72397
Author: arb2196 <arb2196@columbia.edu>
Date:   Thu Oct 23 10:34:14 2014 -0400
commit 04df06ceba966c32a44fb9c95dd067c4f1e3d566
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Oct 22 23:08:01 2014 -0400
commit 47161593fee9bfd68962c67779d4f3183b2ad11e
```

```
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Oct 22 22:29:34 2014 -0400
commit e5804d53c37e2e2f1ef58466884a856336d12f6c
Author: blancgab <gab2135@columbia.edu>
Date:   Thu Oct 16 00:42:46 2014 -0400
commit 73fe669b9183b0ef3a0e7d0ca0fc9a3532a44c27
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Oct 15 19:38:46 2014 -0400
commit 3c2eb93c7eb1952349ff7e767a85649378d01b16
Author: Gabe Blanco <gab2135@columbia.edu>
Date:   Wed Sep 24 23:17:31 2014 -0400
commit 40281bf6f7075845d7c84f3e5c1608dd338280b7
Author: Gabe Blanco <gab2135@columbia.edu>
Date:   Wed Sep 24 23:16:35 2014 -0400
commit 5e753b5c18bda11caec8ff5591e00e6e7f40e194
Author: Gabe Blanco <gab2135@columbia.edu>
Date:   Wed Sep 24 23:16:18 2014 -0400
commit eb895bd1e4df461e01d475faad1fbcc1a24888cb
Author: Gabe Blanco <gab2135@columbia.edu>
Date:   Wed Sep 24 23:15:17 2014 -0400
commit f1cb903628729eb71880921220f0592168a72286
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Sep 24 22:56:00 2014 -0400
commit c4395e2f0fc63dfd3549d0ddf418e0e44b57a5ad
Author: arb2196 <arb2196@columbia.edu>
Date:   Wed Sep 24 22:54:10 2014 -0400
commit 498663e568639349d1f04a5be190efaf148c0e5e
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Sep 24 22:38:56 2014 -0400
commit 0c345b7c304d430e39e7c1e4747c3bf41d3aa26b
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Wed Sep 24 15:17:28 2014 -0400
commit ed35d56828eb1aee721cf4713ead14ed34cb2136
Author: Benjamin Barg <benbakerbarg@gmail.com>
Date:   Tue Sep 23 20:14:41 2014 -0400
```

5. Architectural Design

5.1. Block Diagram



5.2. Preprocessor

Filename: `preprocessor.py`

A preprocessor written in Python that takes the whitespace delimited `*.sht` source code and produces an output that can be tokenized by the scanner. The preprocessor goes line by line and calculates where new blocks of code are created based on the indentation level, as well as replacing newlines with semi-colons to mark the end of statements (unless a linejoin \ character is used). Inline comments and block comments are also removed in the preprocessor; this is required to correctly calculate correct indentation levels.

In addition to demarking scope from whitespace, the preprocessor also checks for certain invalid characters (*tabs*, *right-brackets* and *left-brackets*) and throws an error if they are encountered, since they could break the scanner.

The preprocessor is called by the execution script, and produces an intermediary output file called with the extension `*.proc.sht`

Source Code: `example.sht`

```
#~ Code before being preprocessed ~#
int result

func int snuggle():
    int[] output[10]

    for (int i=0; i<10 ; i=i+1;):
        if (x == 9):
            output[i] = 1 ## only last elem=1
        else:
            result[i] = 0

    return result
```

Preprocessed Code: `example.proc.sht`

```
int result;
func int snuggle():{
    int[] output[10];
    for (int i=0; i<10 ; i=i+1;):{
        if (x == 9):{
            output[i] = 1;
}         else:{
            result[i] = 0;
}}    return result;
}
```

5.3. Scanner

Relevant Files: `scanner.mll`

The scanner, written in *OCamlLex*, takes the intermediary output from the preprocessor and tokenizes it into *keywords*, *identifiers* and *literals*. It also removes the rest of the block comments that were not removed in the preprocessor as well as *whitespace* (which is no longer syntactically useful). If there exist any other characters which cannot be lexed by the scanner, or if an *identifier* or *literal* is not syntactically valid, the scanner will throw an error.

The tokens created by the scanner are then used by the Parser to create an *Abstract Syntax Tree*.

5.4. Parser and AST

Relevant Files: `parser.mly, ast.ml`

The parser, written in *OCamlYacc*, takes a series of tokens and then, using the grammar declared in `parser.mly` and the datatypes defined in `ast.ml`, generates an *Abstract Syntax Tree*. In parser.mly we define the grammar using productions and rules. If the code can be successfully parsed, that means it is syntactically (although not necessarily semantically) correct.

5.5. Symbol Table

Relevant Files: `environment.ml`

From the *Abstract Syntax Tree* that is generated by the Parser, we can build a symbol table from all of the identifiers. For each identifier, the symbol table stores information about the type, scope and location of each identifier. By establishing a Symbol Table, the compiler makes certain that an identifier for either a variable or a function does not get rewritten within the same level of scope, and allows us to do type checking of identifier during the Semantic Checking step.

5.6. Semantic Checking and Code Generation

Relevant Files: `generator.ml, generator_utility.ml`

Sheets compilation is single-pass depth-first traversal of the *Abstract Syntax Tree*, which means that *semantic checking and code generation* happen simultaneously. If an expression or statement passes the semantic check, the translated code in OpenCL is appended to the target file, otherwise the compiler halts and throws a compilation error.

In *semantic checking*, the compiler verifies the validity *Abstract Syntax Tree*. For every binary operator, the compiler checks that the two elements of the operation have the same type since *Sheets* does not support type casting. The *Symbol Table* is used to verify that any identifier which gets called has a) previously been declared, b) is in scope and c) is of the correct type.

If a node within the AST passes semantic check, then the compiler generate the equivalent code in *OpenCl*. For statements in standard functions, translation is a fairly straightforward process; however when translating *gfunctions*, the generator needs to interpret them into *OpenCl* kernel files, which is significantly more complex.

6. Test Plan

6.1. Source to Target

Source Program:

```
gfunc float[] filter(float[] a).[1]:

    #~ block size is 1, so Block.start is just our index ~#
    int i = Block.start

    if (a[i] < 5.0):
        Block.out[i] = 0.0
    else:
        Block.out[i] = a[i]

func int snuggle():
   float[] a[5] = [10.0, 20.1, 5.2, 2.3, 90.4]
   float[] result[5] = filter(a)
```

Target Result:

```
#include <stdio.h>
#include "aws-g2.2xlarge.h"
#include "cl-helper.h"
#include "timing.h"
#include <CL/cl.h>

#define time_start() get_timestamp(&start)
#define time_end() get_timestamp(&end)
timestamp_type start;
timestamp_type end;
cl_context __sheets_context;
cl_command_queue __sheets_queue;
cl_int __cl_err;
const char *filter_kernel_string =
"__kernel void filter(__global const int __arr_len, __global double *__out,__global
const double * a){const int _id = get_global_id(0);const int __block_start = _id *
1;const int __block_end = _id * 1 + 1;int i = __block_startif( < 5.)__out[i] = 0.;
} else __out[i] = a[i];}";
const char *filter_kernel_name = "filter";
cl_kernel filter_compiled_kernel;
double * filter(int __arr_len, double * a)
{
cl_mem __arg1 = clCreateBuffer(__sheets_context,
CL_MEM_WRITE_ONLY,
sizeof(double) * __arr_len,
NULL,
&__cl_err);
CHECK_CL_ERROR(__cl_err, "clCreateBuffer");
cl_mem __arg2 = clCreateBuffer(__sheets_context,
CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
sizeof(double) * __arr_len,
(void *) a,
&__cl_err);
CHECK_CL_ERROR(__cl_err, "clCreateBuffer");
CALL_CL_GUARDED(clEnqueueWriteBuffer,
(__sheets_queue,
```

```
__arg2,
CL_TRUE,
0,
sizeof(double) * __arr_len,
(void *) a,
0,
NULL,
NULL));
SET_3_KERNEL_ARGS(filter_compiled_kernel,__arr_len,
__arg1,
__arg2
);
size_t gdims[] = { __arr_len };
CALL_CL_GUARDED(clEnqueueNDRangeKernel,(__sheets_queue,
filter_compiled_kernel,
1,
0,
gdims,
NULL,
0,
NULL,
NULL));
double *__out = (double*) malloc(__arr_len * sizeof(double));
CALL_CL_GUARDED(clEnqueueReadBuffer,
(__sheets_queue,
__out,
CL_TRUE,
0,
sizeof(double) * __arr_len,
(void *) __out,
0,
NULL,
NULL));
CALL_CL_GUARDED(clReleaseMemObject, (__arg1));
CALL_CL_GUARDED(clReleaseMemObject, (__arg2));
return __out;
}
int snuggle(){
float a[]  = { 10., 20.1, 5.2, 2.3, 90.4};
float result[]  = filter(100000, a);
}
int main()
{
create_context_on(SHEETS_PLAT_NAME, SHEETS_DEV_NAME, 0, &__sheets_context,
&__sheets_queue, 0);
filter_compiled_kernel = kernel_from_string(__sheets_context, filter_kernel_string,
filter_kernel_name, SHEETS_KERNEL_COMPILE_OPTS);
snuggle();
CALL_CL_GUARDED(clReleaseKernel, (filter_compiled_kernel));
CALL_CL_GUARDED(clReleaseCommandQueue, (__sheets_queue));
CALL_CL_GUARDED(clReleaseContext, (__sheets_context));
return 0;
}
```

6.2. Test Suites

As we developed the compiler, we wrote tests every time that we implemented a new feature to verify that it works. For each new feature, we wrote a minimum of two tests: One intended to pass, and one intended to fail. We built a lot of small tests to test individual elements, such as simple operators, function calls, for loops, variable declarations, etc.

Then, we also had “black box” tests, large programs with a lot of possible point of failures. The black box tests were the last ones that we managed to get to successfully pass.

```
Running all tests in current directory

  Note: an 'x' next to a test indicates that the test
  is failing when it is expected to pass, or passing
  when expected to fail. This does not guarantee that
  a successful test is passing or failing for the
  expected reason, so make sure to verify that all
  outputs are correct

[ ] Running Test: 001_p_funccall
[ ] Running Test: 002_n_funccall
[ ] Running Test: 003_p_funccall
[ ] Running Test: 004_n_funccall
[ ] Running Test: 005_p_globalsum
[ ] Running Test: 006_p_globalsum
[ ] Running Test: 007_p_globalsum
[ ] Running Test: 008_p_globalsum
[ ] Running Test: 009_p_globalproduct
[ ] Running Test: 010_p_globalproduct
[ ] Running Test: 011_p_globalproduct
[ ] Running Test: 012_p_assignment
[ ] Running Test: 013_n_assignment
[ ] Running Test: 014_p_initialization
[ ] Running Test: 015_p_initialization
[ ] Running Test: 016_p_return
[ ] Running Test: 017_p_return
[ ] Running Test: 018_n_return
[ ] Running Test: 019_n_return
[ ] Running Test: 020_n_return
[ ] Running Test: 021_p_return
[ ] Running Test: 022_p_ifelse
[ ] Running Test: 023_n_ifelse
[ ] Running Test: 024_n_ifelse
[ ] Running Test: 025_p_ifelse
[ ] Running Test: 026_n_ifelse
[ ] Running Test: 027_p_for
[ ] Running Test: 028_p_for
[ ] Running Test: 029_n_for
[ ] Running Test: 030_n_for
[ ] Running Test: 031_n_for
[ ] Running Test: 032_p_for
[ ] Running Test: 033_p_while
[ ] Running Test: 034_n_while
[ ] Running Test: 035_n_while
[ ] Running Test: 036_p_vdecl
[ ] Running Test: 037_n_vdecl
[ ] Running Test: 038_n_fdecl
[ ] Running Test: 039_p_fdecl
[ ] Running Test: 040_p_literals
```

```
[ ] Running Test: 041_n_literals
[ ] Running Test: 043_p_literals
[ ] Running Test: 044_p_literals
[ ] Running Test: 045_n_literals
[ ] Running Test: 046_n_literals
[ ] Running Test: 047_n_arguments
[ ] Running Test: 048_p_gfdecl
[ ] Running Test: 049_n_gfdecl
[ ] Running Test: 050_p_gfdecl
[ ] Running Test: 051_n_gfdecl
[ ] Running Test: 052_n_identifier
[ ] Running Test: 053_p_arrayaccess
[ ] Running Test: 054_n_arrayaccess
[ ] Running Test: 900_BigTest
[ ] Running Test: 901_BigTest2
[ ] Running Test: 902_BigTest3
[ ] Running Test: 903_Filter
done
OUTPUT FILE: 'envtests.output'
```

6.3. Test Automation

With upwards up sixty unique tests in our suite by the end of the project, automation became a necessity very quickly. We wrote a test script, `run_all.sh`, that not only runs every test, but also checks the test name for the string "`_n_`", which indicates a test that is intended to fail in the compiler. By redirecting the `stderr`, the testing script checks to see if a test fails and then prints a result (see above) of whether the test behaves as expected.

If a test fails when it was expected to pass, or vice versa, then the script marks that test with an "`X`" in the output. The script also writes the output, both stdin and stderr, to an output file which the programmer can then read to determine the exact cause of the error.

The testing script is included in the Appendix.

7. Lessons Learned

7.1. Gabriel Blanco

I learned a valuable skill in organizing a group project: coordinating sleep schedules with your group members for maximum efficiency. You need to fine tune it so that when one member of your team is starts to get sleepy, then another one can step in and pick up where the other left off, maximizing *Peak Coding Efficiency Hours* (PCEH). If one member was awake most of the previous night, they'll need to get more sleep the following day. Caffeine adds a whole 'nother wrench into the equation, because that's just one more variable to account for. Further testing is required.

7.2. Amelia Brunner

I learned that setting up concrete timelines for large scale projects takes more effort than just setting deadlines onto the calendar; it requires actually UNDERSTANDING the gravity of tasks involved and planning for large tasks instead of letting them ambush you at the last minute. Also, functional programming is cool.

7.3. Ruchir Khaitan

Having a consistent coding style and naming convention is very important. Block comments above function definitions are critical. Wear sunscreen. Variable and function names should tell a story. Currying is good; curry is better.

7.4. Ben Barg

Don't push code until it you've run it, and made sure it builds and passes all the tests! When you are working on a team project, you have to be constantly conscious of breaking other people's code. Also, make sure to implement the simplest possible functionality of your project first, and only then start adding features; this is the best way to avoid overextending yourself.

Lastly, I learned that functional programming is indeed quite useful.

8. Appendix

8.1. Preprocessor

preprocessor.py

```
#! /usr/bin/python

# Sheets preprocessor
#
# Author: Gabriel Blanco
# Copyright 2014, Symposium Software

import os
import re
import sys

# Find the best implementation available on this platform
try:
  from cStringIO import StringIO
except:
  from StringIO import StringIO

def process(input_file):
  invalidchar = ('{','}','\t')
  blockcomment = ['#~','~#']

  stack = [0]
```

```
output = StringIO()
newindent = False
commented = False
linejoin  = False

for i, line in enumerate(input_file):
  lineout = remove_inline(line)

  if lineout:
    for x in invalidchar:
      if x in lineout:
        error("SyntaxError: Invalid character {} found on line {}".format(x,i))

    # Check if first statement is a block comment
    lstripline = lineout.lstrip()

    if len(lstripline) > 1 and blockcomment[0] == lstripline[:2]:
      commented = True

    # Checks if line gets uncommented
    if commented:
        if len(lineout) > 1 and blockcomment[1] == lineout[-2:]:
          commented = False
    else:

      if not linejoin:
        wcount  = len(lineout) - len(lineout.lstrip(' '))

        # If the previous line began an indentation, add the new
        # indentation level to the block (so long as the new indentation
        # level is greater than the previous one)
        if newindent == True:
          if wcount > stack[-1]:
            stack.append(wcount)
            newindent = False
          else:
            error("IndentationError on line {}".format(i))

        # If the indentation level is greater than expected, throw an error
        if wcount > stack[-1]:
          error("IndentationError on line {}".format(i))

        else:

          # If the indentation level is less than the current level, return
          # to a previous indentation block. Throw an error if you return to
          # an indentation level that doesn't exist
          while(wcount < stack[-1]):
            lineout = "}" + lineout
            stack.pop()

          if wcount != stack[-1]:
            error("IndentationError on line {}".format(i))

      # Given that the indentation level is correct, check for the start
      # of a new code block (where a line ends with a ':') and insert a
      # '{'. At the end of a line, add a semicolon ';' unless if there is
      # a linejoin character '\'.
      if lineout[-1] == ':':
        lineout = lineout + '{\n'
        newindent = True

      elif lineout[-1] == '\\':
        linejoin = True
        lineout = lineout[:-1]

      else:
```

```
          lineout = lineout + ';\n'
          linejoin = False

        output.write(lineout)

  while 0 < stack[-1]:
    output.write("}")
    stack.pop()

  return output

def error(msg):
  sys.stderr.write(msg+"\n")
  sys.exit(2)

def remove_inline(line):
  if "##" in line:
    regex = re.compile("^(.*?)#.*|.*")
    m = regex.match(line)
    comments_removed = m.group(1)
  else:
    comments_removed = line
  return comments_removed.rstrip()

def usage():
  print"""
  python preprocessor.py [input.sht]
  """

if __name__ == "__main__":

  if len(sys.argv) != 2:
    usage()
    sys.exit(2)

  try:
    f_in = open(sys.argv[1],"r")
  except IOError:
    error("IOError: Cannot read input file %s.\n" % sys.argv[1])

  name_ext = os.path.basename(f_in.name)
  dir_ext = os.path.dirname(f_in.name)+"/"

  if name_ext.lower().endswith((".sht",".sheet")):
    fname = os.path.splitext(name_ext)[0]
  else:
    error('NameError: Input must have Sheets file extension')

  out_str = process(f_in)

  f_out = open(dir_ext+fname+".proc.sht", 'w')
  f_out.write(out_str.getvalue())
```

8.2. Scanner

`scanner.mll`

```
(*
 * Sheets scanner
 *
```

```
 * Authors: Gabriel Blanco, Ruchir Khaitan
 * Copyright 2014, Symposium Software
 *)

{ open Parser;; }

let num = ['0'-'9']
let flt = '-'?num+ '.' num* | '.' num+

rule token = parse
(* Whitespace *)
| [' ' '\n' '\r'] { token lexbuf }

(* Comments *)
| "#~" { comment lexbuf }

(* Punctuation *)
| '('         { LPAREN }  | ')'         { RPAREN }
| '{'         { LBRACE }  | '}'         { RBRACE }
| '['         { LBRACK }  | ']'         { RBRACK }
| ';'         { SEMI }    | ','         { COMMA }
| '.'         { PERIOD }  | ':'         { COLON }

(* Arithmetic Operators *)
| '+'         { PLUS }    | '-'         { MINUS }
| '*'         { TIMES }   | '/'         { DIVIDE }

(* Relational Operators *)
| "=="        { EQ }      | "!="        { NEQ }
| '<'         { LT }      | "<="        { LEQ }
| ">"         { GT }      | ">="        { GEQ }

(* Assignment Operator *)
| '='         { ASSIGN }

(* Conditional Keywords *)
| "if"        { IF }      | "else"      { ELSE }

(* Loop Keywords*)
| "while"     { WHILE }   | "for"       { FOR }
| "break"     { BREAK}    | "continue" { CONTINUE }

(* Function Keywords *)
| "func"      { FUNC }    | "gfunc"     { GFUNC }
| "return"    { RETURN }

(* Type Keywords*)
| "int"       { INT }     | "float"     { FLOAT }
| "Block"     { BLOCK }

(* End-of-File *)
| eof { EOF }

(* Identifiers *)
| ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }

(* Literals *)
| '-'?num+ as intlit { INT_LITERAL(int_of_string intlit) }
| flt      as fltlit { FLOAT_LITERAL(float_of_string fltlit) }

(* Throw Error for Invalid Token *)
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
| "~#" { token lexbuf }      (* End-of-comment *)
| _    { comment lexbuf }    (* Eat everything else *)
```

8.3. Parser

parser.mly

```
/*
 * Sheets parser
 *
 * Authors: Amelia Brunner, Gabriel Blanco, Ben Barg
 * Copyright 2014, Symposium Software
 */

%{ open Ast;; %}


////////////////////////////////////////////////////////////////////////////////
///////////////////////////// TOKEN DECLARATIONS ///////////////////////////////
////////////////////////////////////////////////////////////////////////////////


/* Punctuation Tokens */
%token LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK SEMI COMMA PERIOD COLON EOF

/* Loop Keywords */
%token WHILE FOR BREAK CONTINUE

/* Conditional Keywords */
%token IF ELSE

/* Function Keywords */
%token FUNC GFUNC BLOCK RETURN

/* Type Keywords */
%token INT FLOAT

/* Operator Tokens */
%token PLUS MINUS TIMES DIVIDE
%token EQ NEQ LT LEQ GT GEQ

/* Assignment Operator */
%token ASSIGN

%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <int list> INT_ARRAY_LITERAL
%token <float list> FLOAT_ARRAY_LITERAL
%token <string> ID

/* Precedence Definition */
%nonassoc NOELSE ELSE
%right ASSIGN G_ASSIGN
%left EQ NEQ
%left LT LEQ GT GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left PERIOD

%start program                          /* start symbol */
%type <Ast.program> program             /* type returned by program */

%%

////////////////////////////////////////////////////////////////////////////////
```

```
///////////////////////////// START SYMBOL //////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


program:                        /* [vdecls], [fdecls] */
    | /* Empty Program */       { [], [] }
    | program vdecl SEMI        { ($2 :: fst $1), snd $1 }
    | program fdecl             { fst $1, ($2 :: snd $1) }
    | program gfdecl            { fst $1, ($2 :: snd $1) }


/////////////////////////////////////////////////////////////////////////////
////////////////////////////// FUNCTIONS ////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


/* func <type> <id>(arg1, arg2, ...):{ <statements> }; */
fdecl:
    FUNC type_name ID LPAREN formals_opt RPAREN COLON
    LBRACE stmt_list_opt RBRACE
    {{
        r_type    = $2;                 (* return type *)
        fname     = $3;                 (* function name *)
        formals   = $5;                 (* list of arguments *)
        body      = $9;                 (* statement list *)
        isGfunc   = false;              (* false b/c not a gfunc *)
        blocksize = -1                  (* block size unused *)
    }}

/* gfunc <type> <id>(arg1, arg2, ...).[<blocksize>]:{ <statements> }; */
gfdecl:
    GFUNC type_name ID LPAREN formals_opt RPAREN blocksize COLON
    LBRACE gfunc_stmt_list_opt RBRACE
    {{
        r_type    = $2;                 (* return type *)
        fname     = $3;                 (* gfunction name *)
        formals   = $5;                 (* list of arguments *)
        body      = $10;                (* gfunction statement list *)
        isGfunc   = true;               (* true b/c a gfunc *)
        blocksize = $7                  (* block size *)
    }}

/* Optional Formal Arguments */
formals_opt:
    | /* Nothing */                      { [] }
    | formal_list                        { List.rev $1 }

formal_list:
    | vdecl                              { [$1] }
    | formal_list COMMA vdecl            { $3 :: $1 }

/* Blocksize in Gfunction Definition */
blocksize:
    | /* Nothing */                      { 1 }
    | PERIOD LBRACK INT_LITERAL RBRACK    { $3 }


/////////////////////////////////////////////////////////////////////////////
////////////////////////////// VARIABLES ////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


/* Type Declaration */
type_name:
    | INT LBRACK RBRACK                  { "int[]" }
    | FLOAT LBRACK RBRACK                { "float[]" }
    | INT                                { "int" }
```

```
    | FLOAT                               { "float" }

/* Optional Array Size Declaration */
array_opt:
    | /*Nothing*/                         { -1 }
    | LBRACK INT_LITERAL RBRACK           { $2 }

/* <type> name [<array_size>] */
vdecl:
    type_name ID array_opt
    {{
        v_type   = $1;                   (* variable type *)
        v_name   = $2;                   (* variable name *)
        a_size   = $3;                   (* array size *)
    }}

vdecl_list:
    | vdecl SEMI                          { [$1] }
    | vdecl_list vdecl SEMI               { $2 :: $1 }


//////////////////////////////////////////////////////////////////////////////
////////////////////////////// STATEMENTS ////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////


/* Optional List of Statements */
stmt_list_opt:
    | /* Nothing */                       { [] }
    | stmt_list                           { List.rev $1 }

stmt_list:
    | stmt                                { [$1] }
    | stmt_list stmt                      { $2 :: $1 }

/* Optional List of Gfunction Statements */
gfunc_stmt_list_opt:
    | /* Nothing */                       { [] }
    | gfunc_stmt_list                     { List.rev $1 }

gfunc_stmt_list:
    | gstmt                               { [$1] }
    | gfunc_stmt_list gstmt               { $2 :: $1 }

/* Optional List of Arguments */
args_opt:
    | /* Nothing */                       { [] }
    | args_list                           { List.rev $1 }

args_list:
    | expr                                { [$1] }
    | args_list COMMA expr                { $3 :: $1 }

/* Statements are found in the body of a functions, Gstatements
 * are found in Gfunctions. Gstatements contail all statements
 * as well as access to the special Block construct. */
stmt:
    | vdecl SEMI                                        { Vdecl($1) }
    | expr SEMI                                         { Expr($1) }
    | RETURN expr SEMI                                  { Return($2) }
    | assign_expr ASSIGN expr SEMI                      { Assign($1, $3) }
    | vdecl ASSIGN expr SEMI                            { Init($1, $3) }
    | LBRACE stmt_list RBRACE                           { Block(List.rev $2) }
    | IF bool_block COLON block_body %prec NOELSE       { If($2, $4, Block[] ) }
    | IF bool_block COLON block_body ELSE COLON block_body    { If($2, $4, $7) }
    | FOR for_pt1 for_pt2 for_pt3 COLON block_body      { For($2, $3, $4, $6) }
    | WHILE bool_block COLON block_body                 { While($2, $4) }
```

```
gstmt:
    | vdecl SEMI                                         { Vdecl($1) }
    | expr SEMI                                          { Expr($1) }
    | blockexpr SEMI                                     { Expr($1) }
    | g_assign_expr ASSIGN expr SEMI                     { Assign($1, $3) }
    | g_assign_expr ASSIGN blockexpr SEMI                { Assign($1, $3) }
    | vdecl ASSIGN expr SEMI                             { Init($1, $3) }
    | vdecl ASSIGN blockexpr SEMI                        { Init($1, $3) }
    | IF bool_block COLON gblock_body %prec NOELSE       { If($2, $4, Block([])) }
    | IF bool_block COLON gblock_body ELSE COLON gblock_body  { If($2, $4, $7) }
    | FOR gfor_pt1 gfor_pt2 gfor_pt3 COLON gblock_body  { For($2, $3, $4, $6) }
    | WHILE bool_block COLON gblock_body                 { While($2, $4) }

/* Conditional and Loop Statements*/
bool_block: LPAREN bool_expr RPAREN                      { $2 }

block_body:  LBRACE loop_stmt_list RBRACE                { Block(List.rev $2) }
gblock_body: LBRACE gloop_stmt_list RBRACE               { Block(List.rev $2) }

for_pt1: LPAREN stmt                                     { $2 }
for_pt2: bool_expr SEMI                                  { $1 }
for_pt3: stmt RPAREN                                     { $1 }

gfor_pt1: LPAREN gstmt                                   { $2 }
gfor_pt2: bool_expr SEMI                                 { $1 }
gfor_pt3: gstmt RPAREN                                   { $1 }

/* Loops can contain all normal expressions, and also Break and Continues */
loop_stmt_list:
    | /* Nothing */                                      { [] }
    | stmt_list                                          { $1 }
    | loop_stmt_list loopexpr                            { $2 :: $1 }

gloop_stmt_list:
    | /* Nothing */                                      { [] }
    | gfunc_stmt_list                                    { $1 }
    | gloop_stmt_list loopexpr                           { $2 :: $1 }


/////////////////////////////////////////////////////////////////////////////
//////////////////////////// EXPRESSIONS //////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


loopexpr:
    | CONTINUE SEMI                   { Continue }
    | BREAK SEMI                      { Break }

blockexpr:
    | BLOCK PERIOD ID                                { BlockAcc($3, Literal_int(-1)) }
    | BLOCK PERIOD ID LBRACK INT_LITERAL RBRACK  { BlockAcc($3, Literal_int($5)) }
    | BLOCK PERIOD ID LBRACK ID RBRACK           { BlockAcc($3, Id($5)) }

bool_expr:
    | expr EQ expr                    { Binop($1, Equal,   $3) }
    | expr NEQ expr                   { Binop($1, Neq,     $3) }
    | expr LT expr                    { Binop($1, Less,    $3) }
    | expr LEQ expr                   { Binop($1, Leq,     $3) }
    | expr GT expr                    { Binop($1, Greater, $3) }
    | expr GEQ expr                   { Binop($1, Geq,     $3) }

array_expr:
    | ID                              { Id($1) }
    | array_literal                   { $1 }

assign_expr:
```

```
    | array_expr LBRACK expr RBRACK   { ArrayAcc($1, $3) }
    | ID                             { Id($1) }

g_assign_expr:
    | blockexpr                      { $1 }
    | assign_expr                    { $1 }

/* Literals */
literal:
    | INT_LITERAL                    { Literal_int($1) }
    | FLOAT_LITERAL                  { Literal_float($1) }
    | array_literal                  { $1 }

array_literal:
    | LBRACK int_literal_list RBRACK          { Literal_int_a(List.rev $2) }
    | LBRACK float_literal_list RBRACK        { Literal_float_a(List.rev $2) }

int_literal_list:
    | INT_LITERAL                             { [$1] }
    | int_literal_list COMMA INT_LITERAL      { $3 :: $1 }

float_literal_list:
    | FLOAT_LITERAL                           { [$1] }
    | float_literal_list COMMA FLOAT_LITERAL  { $3 :: $1 }

/* Expressions */
expr:
    | literal                        { $1 }
    | ID LPAREN args_opt RPAREN      { Call($1, $3) }
    | array_expr LBRACK expr RBRACK  { ArrayAcc($1, $3) }
    | ID                             { Id($1) }
    | bool_expr                      { $1 }
    | expr PLUS expr                 { Binop($1, Plus, $3) }
    | expr MINUS expr                { Binop($1, Minus, $3) }
    | expr TIMES expr                { Binop($1, Times, $3) }
    | expr DIVIDE expr               { Binop($1, Divide, $3) }
    | LPAREN expr RPAREN             { $2 }
```

8.4. AST

ast.ml

```
(*
 * Sheets Abstract Syntax Tree types
 *
 * Authors: Amelia Brunner, Gabriel Blanco
 * Copyright 2014, Symposium Software
 *)

type op = Plus    | Minus   | Times   | Divide  | Equal   |
          Neq     | Less    | Leq     | Greater | Geq

type expr =
    | Literal_int of int
    | Literal_int_a of int list
    | Literal_float of float
    | Literal_float_a of float list
    | Id of string
    | Binop of expr * op * expr
    | Call of string * expr list
    | ArrayAcc of expr * expr
    (*   ArrayAcc(expr1,expr2)
     *     expr1  : evaluates to an array,
```

```
     *     expr2  : evaluates to the index of the element to be accessed *)
    | BlockAcc of string * expr
    (*   BlockAcc(string,expr)
     *     string : field of block (i.e. "start", "end", "out")
     *     expr   : index of block.out (Literal_int(0) otherwise *)

type vdecl = {
    v_type     : string;    (* PARSER *)
    v_name     : string;
    a_size     : int;
}

(* Statements *)
type stmt =
    | Vdecl of vdecl
    | Block of stmt list
    | Expr of expr
    | Assign of expr * expr
    | Return of expr
    | Init of vdecl * expr
    | If of expr * stmt * stmt
    | For of stmt * expr * stmt * stmt
    | While of expr * stmt
    | Continue
    | Break

(* Function Declaration *)
type fdecl = {
    r_type     : string;
    fname      : string;
    formals    : vdecl list;
    body       : stmt list;
    isGfunc    : bool;
    blocksize  : int;
}

type program = vdecl list * fdecl list

type datatype =
    | Int
    | Float
    | Array of datatype
```

8.5. Generator

```
(*
 * Sheets Code Generator
 *
 * Authors: Ruchir Khaitan, Ben Barg, Amelia Brunner
 * Copyright 2014, Symposium Software
 *)

open Ast;;
open Environment;;
open Printf;;
open String;;

exception SyntaxError of int * int * string;;
exception NotImplementedError of string;;
exception UndefinedTypeError;;
exception BadExpressionError of string;;
```

```
(* ------------------------------------------------------------------ *)
(* Process Statements and expressions                                 *)
(* ------------------------------------------------------------------ *)
(* Strategy for syntax checking:
   generate_expr will be a set of case matchings that will
   call gen_checked_<EXPRESSION_NAME> that can meaningfully be
   checked.

   gen_checked_<EXPRESSION_NAME> will take a function <CHECKER> that
   returns a boolean an expression env.  <CHECKER> can be in
   Environment as it is for variable ids <CHECKER> takes an expression
   and env *)
let rec generate_type datatype env =
  match datatype with
  | Int ->        Environment.append env [Text("int")]
  | Float ->      Environment.append env [Text("double")]
  | Array(t) ->   Environment.append env [Generator(generate_type t);
                                   Text("*")] (* Handling array types differently *)

let generate_checked_id check_id id env =
    if (check_id id env) then
        Environment.append env [Text(id)]
    else raise (VariableNotFound id)

let op_to_txt op =
  match op with
  | Plus -> "+"
  | Minus -> "-"
  | Times -> "*"
  | Divide -> "/"
  | Equal -> "=="
  | Greater -> ">"
  | Less -> "<"
  | Geq -> ">="
  | Leq -> "<="
  | Neq -> "!="
  | _-> ""

let rec exp_to_txt exp =
  match exp with
    Literal_int(i) -> string_of_int(i)
  | Literal_float(f) -> string_of_float(f)
  | Id(s) -> s
  | Binop(e1, op, e2) -> (exp_to_txt e1) ^ " " ^ (op_to_txt op) ^ " " ^ (exp_to_txt e2)
  | Literal_string(s) -> "\"" ^ s ^ "\""
  | _-> ""

let rec args_to_txt arg_list str=
  match arg_list with
  | [] ->
     if(String.contains str ',') then
       String.sub str 0 (String.length str - 2)
     else
       str
  | arg :: arg_tail -> args_to_txt arg_tail (str ^ (exp_to_txt arg) ^ ", ")

let generate_checked_binop check_binop binop env =
  check_binop binop env;
  match binop with
    Binop(e1, op , e2) ->   Environment.append env [
                                          Text((exp_to_txt e1) ^ " "
                                               ^ (op_to_txt op) ^ " " ^
                                                (exp_to_txt e2))]
  | _->  raise (BadExpressionError("binop"))

let generate_checked_array_access check_array_access array_expr env =
```

```
  check_array_access array_expr env;
  match array_expr with
  | ArrayAcc(e1, e2) ->       Environment.append env [
                                            Text((exp_to_txt e1) ^ "[" ^
                                                  (exp_to_txt e2) ^ "]")]
  | _-> raise (BadExpressionError("Array Access"))

let generate_checked_f_call check_f_call f_call env =
  check_f_call f_call env;
  match f_call with
  | Call(id, expressions) ->
     if Environment.id_is_gfunc id env then
       Environment.append env [Text(id ^ "(100000, "
                               ^ (args_to_txt expressions "")
                                    ^ ")");]
     else
       Environment.append env [Text(id ^ "("
                               ^ (args_to_txt expressions "")
                               ^ ")")]
  | _-> raise (BadExpressionError("Function Call"))

let rec print_int_array array_list str =
  match array_list with
  | [] ->
     if(String.contains str ',') then
       String.sub str 0 (String.length str - 2)
     else
       str
  | head :: array_tail ->
     print_int_array array_tail (str ^ (string_of_int head) ^ ", ")

let rec print_float_array array_list str =
  match array_list with
  | [] ->
     if(String.contains str ',') then
       String.sub str 0 (String.length str - 2)
     else
       str
  | head :: array_tail ->
     print_float_array array_tail (str ^
                                (string_of_float head) ^ ", ")

let rec generate_exp exp env =
  match exp with
  | Literal_int(i) ->     Environment.append env [Text(string_of_int(i))]
  | Literal_float(f) ->   Environment.append env [Text(string_of_float(f))]
  | Literal_int_a(list_i) -> Environment.append env [Text("{ ");
                                              Text(print_int_array list_i "");
                                              Text("}")]
  | Literal_float_a(list_f) -> Environment.append env [Text("{ ");
                                                Text(print_float_array list_f "");
                                                Text("}")]
  | Id(s) ->          Environment.append env [Generator(generate_checked_id
is_var_in_scope s )]
  | Binop(_,_,_) ->   Environment.append env [Generator(generate_checked_binop
                                                  Generator_utilities.expr_typeof exp)]
  | Call(func_id, formals_list) ->
     Environment.append env [Generator(generate_checked_f_call
                                   Generator_utilities.expr_typeof exp)]
  | ArrayAcc(_, _) -> Environment.append env [Generator(generate_checked_array_access
                                                  Generator_utilities.expr_typeof exp)]
  | BlockAcc(id, exp) ->Environment.append env [Generator(generate_checked_block id
exp)]
  | _-> raise (NotImplementedError("unsupported expression"))
and generate_checked_block id exp env =
  match id with
  | "start" -> Environment.append env [Text("__block_start")]
```

```
  | "end" -> Environment.append env [Text("__block_end")]
  | "out" ->
     match exp with
     | Literal_int(a) ->
        if(a = -1) then
          raise (BadExpressionError("Invalid block access"))
        else
          Environment.append env [Text("__out[" ^ (string_of_int a) ^ "]")]
     | _ -> Environment.append env [Text("__out"); Generator(generate_exp exp)]
     | _-> raise (BadExpressionError("Invalid block access"))
;;


let generate_init vdecl exp env =
  if((Generator_utilities.vdecl_type vdecl) = (Generator_utilities.expr_typeof
                                          exp env)) then
    let v_type = Generator_utilities.vdecl_type vdecl in
    match v_type with
      Array(Int) ->  Environment.append env [Env(add_var vdecl.v_name (v_type));
                                             Text("int" ^ " " ^ vdecl.v_name ^ "[]  =
");

                                             Generator(generate_exp exp);]
    | Array(Float) ->  Environment.append env [Env(add_var vdecl.v_name (v_type));
                                               Text("float" ^ " " ^ vdecl.v_name ^ "[]
= ");

                                               Generator(generate_exp exp);]
    | _-> Environment.append env [Env(add_var vdecl.v_name
(Generator_utilities.vdecl_type vdecl));
                             Text((Generator_utilities.c_type_from_str vdecl.v_type)
                                  ^ " " ^ vdecl.v_name ^ " = ");
                             Generator(generate_exp exp)]
  else
    raise(BadExpressionError("Assignment of incompatible types"))

let generate_return exp env =
    let func_info = Environment.get_func_info env.current_function env in
    let return_type = func_info.return in
    let exp_type = Generator_utilities.expr_typeof exp env in
    if(exp_type = return_type) then
        Environment.append env [
        Text("return ");
        Generator(generate_exp exp)]
    else
        raise(BadExpressionError("Bad return type"))

let generate_assign id exp env =
    match id with
    | Id(a) -> if (is_var_in_scope a env) then
                 if(Generator_utilities.expr_typeof id env =
                    Generator_utilities.expr_typeof exp env) then
                       Environment.append env [
                       Text(a ^ " =" ); Generator(generate_exp exp)]
                 else
                       raise(BadExpressionError("Assignment of incompatible
                       types"))
              else
                   raise (BadExpressionError("assignment to undefined id"))
    | BlockAcc(s, expr) -> Environment.append env [Text("__out[");
                                                  Generator(generate_exp expr);
                                                  Text("] = ");
                                                  Generator(generate_exp exp);
                                                  Text(";")]
    | _-> raise (BadExpressionError("Invalid Assignment"))

let rec process_stmt_list stmt_list env =
   match stmt_list with
   stmt :: other_stmts -> Environment.append env [Generator(process_stmt
```

```
  stmt); Generator(process_stmt_list other_stmts)]
  | []     -> Environment.append env [ Text("") ]
and process_stmt stmt env =
  match stmt with
  Vdecl(vdecl) ->          Environment.append env [
                           Generator(process_vdecl vdecl);
                           Text(";\n") ]
  | Block(stmt_list) ->    Environment.append env [
                           Generator(process_stmt_list stmt_list) ]
  | Expr(expr) ->          Environment.append env [
                           Generator(generate_exp expr );
                           Text(";\n") ]
  | Assign(name, expr) ->  Environment.append env [
                           Generator(generate_assign name expr);
                           Text(";\n")]
  | Return(expr) ->        Environment.append env [
                           Generator(generate_return expr);
                           Text(";\n")]
  | Init(vdecl, expr) ->   Environment.append env [
                           Generator(generate_init vdecl expr);
                           Text(";\n")]
  | If(boolexpr, ifstmt, elsestmt) ->
                           Environment.append env [
                           Generator(generate_if boolexpr ifstmt elsestmt)]
  | While(expr, body) ->   Environment.append env [
                           NewScope(generate_while expr body)]
  | For(s1, e2, s3, body) ->
                           Environment.append env [
                           NewScope(generate_for s1 e2 s3 body)]
  | Continue ->            Environment.append env [
                           Text("continue;\n")]
  | Break ->               Environment.append env [
                           Text("break;\n")]
  | _ -> raise (NotImplementedError("Undefined type of expression"))
and process_vdecl vdecl env =
  let v_datatype = Generator_utilities.str_to_type vdecl.v_type in
  Environment.append env [Env(add_var vdecl.v_name v_datatype);
                         Text((Generator_utilities.c_type_from_str vdecl.v_type)
                          ^ " " ^ vdecl.v_name)]
 and generate_while bool_expr body env =
     match bool_expr with
     | Binop(e1, o, e2) ->
              match o with
                 | Equal -> append_while bool_expr body env
                 | Neq -> append_while bool_expr body env
                 | Greater -> append_while bool_expr body env
                 | Less -> append_while bool_expr body env
                 | Geq -> append_while bool_expr body env
                 | Leq -> append_while bool_expr body env
                 | _-> raise (BadExpressionError ("Binop is not boolean"))
     | _-> raise(BadExpressionError ("Conditional expression is not binop"))
and generate_for stmt1 bool_expr stmt2 body env =
     match bool_expr with
     | Binop(e1, o, e2) ->
             match o with
                 | Equal -> append_for stmt1 bool_expr stmt2 body env
                 | Neq -> append_for stmt1 bool_expr stmt2 body env
                 | Greater -> append_for stmt1 bool_expr stmt2 body env
                 | Less -> append_for stmt1 bool_expr stmt2 body env
                 | Geq -> append_for stmt1 bool_expr stmt2 body env
                 | Leq -> append_for stmt1 bool_expr stmt2 body env
                 | _-> raise (BadExpressionError ("Binop is not boolean"))
     | _-> raise(BadExpressionError ("Conditional expression is not binop"))
 and generate_if bool_expr ifbody elsebody env =
     match bool_expr with
     | Binop(e1, o, e2) ->
             match o with
```

```
                  | Equal -> append_if_else bool_expr ifbody elsebody env
                  | Neq -> append_if_else bool_expr ifbody elsebody env
                  | Greater -> append_if_else bool_expr ifbody elsebody env
                  | Less -> append_if_else bool_expr ifbody elsebody env
                  | Geq -> append_if_else bool_expr ifbody elsebody env
                  | Leq -> append_if_else bool_expr ifbody elsebody env
                  | _-> raise (BadExpressionError ("Binop is not boolean"))
      | _-> raise(BadExpressionError ("Conditional expression is not binop"))
and append_while bool_expr body env =
    Generator_utilities.expr_typeof bool_expr env;
    Environment.append env[Text("While("); Generator(generate_exp bool_expr);
    Text("){\n"); Generator(process_stmt body); Text("}\n")]
and append_if_else bool_exp ifbody elsebody env =
    Generator_utilities.expr_typeof bool_exp env;
    Environment.append env [Text("if("); Generator(generate_exp bool_exp);
    Text("){\n"); NewScope(process_stmt ifbody); Text("\n} else {\n");
    NewScope(process_stmt elsebody); Text("}\n")]
(* For loops have to have assignment, boolean expression,
   assignment *)
and print_in_for_loop stmt first env =
    match stmt with
     Assign(name, expr) ->  if first then Environment.append env [
                            Generator(generate_assign name expr);
                            Text(";")]
                            else  Environment.append env [
                            Generator(generate_assign name expr);]
      | _-> raise (BadExpressionError("Argument in for loop invalid"))
and append_for stmt1 bool_exp stmt2 body env =
    Generator_utilities.expr_typeof bool_exp env;
    Environment.append env [Text("for("); Generator(print_in_for_loop stmt1 true );
    Generator(generate_exp bool_exp); Text("; ");
    Generator(print_in_for_loop stmt2 false); Text("){\n"); Generator(process_stmt
body);
    Text("}\n")]

(* ------------------------------------------------------------------ *)
(* Global Variable Declarations                                        *)
(* ------------------------------------------------------------------ *)

let rec generate_global_vdecl_list vdecls env =
  let generate_global_vdecl vdecl env =
    let v_datatype = Generator_utilities.str_to_type vdecl.v_type in
    Environment.append env [Env((add_var vdecl.v_name v_datatype));
                        Generator(generate_type v_datatype);
                        Text(" " ^ vdecl.v_name ^ ";\n")]
  in
  match vdecls with
    [] -> "", env
  | [vdecl] -> generate_global_vdecl vdecl env
  | vdecl :: other_vdecls ->
     Environment.append env [Generator(generate_global_vdecl vdecl);
                         Generator(generate_global_vdecl_list other_vdecls)]

let rec generate_formals_vdecl_list vdecl_list env =
  let generate_formals_vdecl vdecl env =
    let v_datatype = Generator_utilities.str_to_type vdecl.v_type in
    Environment.append env [Env((add_var vdecl.v_name v_datatype));
                        Generator(generate_type v_datatype);
                        Text(" " ^ vdecl.v_name ^ ", ")]
  in
  match vdecl_list with
    [] -> "", env
  | [vdecl] -> Environment.append env [Env((add_var vdecl.v_name
                                      (Generator_utilities.vdecl_type vdecl)));
                                  Text((Generator_utilities.c_type_from_str
vdecl.v_type)
                                      ^ " " ^ vdecl.v_name)]
```

```
  | vdecl :: other_vdecls ->
     Environment.append env [Generator(generate_formals_vdecl vdecl);
                         Generator(generate_formals_vdecl_list other_vdecls)]

(* ------------------------------------------------------------------ *)
(* Kernel invocation declarations                                      *)
(* ------------------------------------------------------------------ *)
(* If we have a gfunc declared as

   > gfunc float[] my_gfunc(float[] arg1, float[] arg2, int arg3)

   then our generated c code will call this function in the same
   way that cpu functions are called:

   > result = my_gfunc(arg1, arg2, arg3, arg4)

   However, when my_gfunc is actually defined in the c file, its
   contents will be the boilerplate OpenCL code that invokes the
   kernel on the gpu.

   At this point, all semantic checking has been completed, so we
   don't need to worry about checking the function map or anything
   like that.

   NOTES:
   - Because each kernel invocation function has its own C scope, we
     don't have to worry about variable name collision
   - The definitions of these functions will appear interspersed with
     cpu func definitions, but this will not interfere with namespace
     conventions. Essentially, the cpu code thinks it's calling
     another cpu function, but internally that cpu function is
     implemented as a gpu function

   - This method has the side-benefit that we don't have to process
     literals passed to functions differently

   INVARIANTS:
   - the output array and input arrays of an individual gfunc MUST be
     the same size
   - if there are non-array arguments, we rename them to __argn
   - the actual args list starts at arg2 (first 2 are reserved for
     size and output array*)
let generate_kernel_invocation_function fdecl env =
  let base_r_type = Generator_utilities.arr_type_str_to_base_type fdecl.r_type in
  let generate_cl_arg_list fdecl env =
    let rec generate_cl_args arg_n formals env =
      let generate_cl_arg arg_n formal env =
       if Generator_utilities.is_array_type formal.v_type then
         (* create a cl memory buffer for array args *)
         Environment.append env [Text(sprintf "cl_mem __arg%d = clCreateBuffer("
arg_n);
                             Text("__sheets_context,\n");
                             Text("CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,\n");
                             Text(sprintf "sizeof(%s) * __arr_len,\n" base_r_type);
                             Text(sprintf "(void *) %s,\n" formal.v_name);
                             Text("&__cl_err);\n");
                             Text("CHECK_CL_ERROR(__cl_err, \"clCreateBuffer\");\n")]
       else
         (* pass primitives directly *)
         Environment.append env [Text(sprintf "%s __arg%d = %s;\n"
                                        base_r_type arg_n formal.v_name)]
      in
      match formals with
       [] -> "", env
      | formal :: other_formals ->
        Environment.append env [Generator(generate_cl_arg arg_n formal);
                            Generator(generate_cl_args (arg_n + 1) other_formals)]
```

```
    in
    (* we make a cl_mem buffer for the return array first *)
    Environment.append env [Text("cl_mem __arg1 = clCreateBuffer(");
                          Text("__sheets_context,\n");
                          Text("CL_MEM_WRITE_ONLY,\n");
                          Text(sprintf "sizeof(%s) * __arr_len,\n" base_r_type);
                          Text("NULL,\n");
                          Text("&__cl_err);\n");
                          Text("CHECK_CL_ERROR(__cl_err, \"clCreateBuffer\");\n");
                          (* user-defined args start at 2 *)
                          Generator(generate_cl_args 2 fdecl.formals)]
  in
  let generate_cl_enqueue_write_buffer_list fdecl env =
    let rec generate_cl_enqueue_write_buffers arg_n formals env =
      let generate_cl_enqueue_write_buffer arg_n formal env =
       if Generator_utilities.is_array_type formal.v_type then
         Environment.append env [Text("CALL_CL_GUARDED(clEnqueueWriteBuffer,\n");
                                 Text("(__sheets_queue,\n");
                                 Text(sprintf "__arg%d,\n" arg_n);
                                 Text("CL_TRUE,\n"); (* ensure blocking write *)
                                 Text("0,\n");            (* no offset *)
                                 Text(sprintf "sizeof(%s) * __arr_len,\n" base_r_type);
                                 Text(sprintf "(void *) %s,\n" formal.v_name);
                                 Text("0,\n");
                                 Text("NULL,\n");    (* no wait list *)
                                 Text("NULL)");
                                 Text(");\n")]
       else "", env (* no need to alloc buffer for primitives *)
      in
      match formals with
       [] -> "", env
      | formal :: other_formals ->
        Environment.append env [Generator(generate_cl_enqueue_write_buffer arg_n
formal);
                               Generator(generate_cl_enqueue_write_buffers
                                          (arg_n + 1) other_formals)]
    in
    (* user-defined args start at 2 *)
    Environment.append env [Generator(generate_cl_enqueue_write_buffers 2
fdecl.formals)]
  in
  let generate_cl_set_kernel_args fdecl env =
    (* list of __argn vars *)
    let generate_arg_ns num_user_args env =
      let rec _helper num_arg_ns_left arg_n env =
       match num_arg_ns_left with
         0 -> "", env
       | 1 -> Environment.append env [Text(sprintf "__arg%d\n" arg_n)]
       | _ -> Environment.append env [Text(sprintf "__arg%d,\n" arg_n);
                                      Generator(_helper (num_arg_ns_left - 1) (arg_n +
1))]

      in
      Environment.append env [Generator(_helper num_user_args 2)]
    in
    (* only need to add 1 because __arr_len is already in formals list *)
    Environment.append env [Text(sprintf "SET_%d_KERNEL_ARGS(" ((List.length
fdecl.formals) + 2));
                          Text(sprintf "%s_compiled_kernel," fdecl.fname);
                          Text("__arr_len,\n");
                          Text("__arg1,\n");
                          Generator(generate_arg_ns (List.length fdecl.formals));
                          Text(");\n")]
  in
  let global_work_items = function
      (* provide a buffer for when block_size doesn't divide array
         size*)
```

```
      1 -> "__arr_len"
    | n -> "__arr_len /" ^ string_of_int n ^ " + 1"
  in
  let generate_cl_enqueue_nd_range_kernel fdecl env =
    Environment.append env [Text(sprintf "size_t gdims[] = { %s };\n"
                                (global_work_items fdecl.blocksize));
                        Text("CALL_CL_GUARDED(clEnqueueNDRangeKernel,");
                        Text("(__sheets_queue,\n");
                        Text(sprintf "%s_compiled_kernel,\n" fdecl.fname);
                        Text("1,\n"); (* only 1 dimensional array support *)
                        Text("0,\n"); (* 0 offset *)
                        Text("gdims,\n");
                        Text("NULL,\n");
                        Text("0,\n");
                        Text("NULL,\n");
                        Text("NULL)");
                        Text(");\n")]
  in
  let generate_cl_enqueue_read_buffer fdecl env =
    (* only one buffer to read, since there's only one output arg *)
    Environment.append env [Text(sprintf "%s *__out = (%s*) malloc(__arr_len *
sizeof(%s));\n" base_r_type base_r_type base_r_type);
                        Text("CALL_CL_GUARDED(clEnqueueReadBuffer,\n");
                        Text("(__sheets_queue,\n");
                        Text("__out,\n");
                        Text("CL_TRUE,\n"); (* blocking read *)
                        Text("0,\n");    (* 0 offset *)
                        Text(sprintf "sizeof(%s) * __arr_len,\n" base_r_type);
                        Text("(void *) __out,\n");
                        Text("0,\n"); (* empty wait queue *)
                        Text("NULL,\n");
                        Text("NULL)");
                        Text(");\n")]
  in
  let generate_cl_release_list fdecl env =
    let rec generate_cl_releases arg_n formals env =
      let generate_cl_release arg_n formal env =
       (* only release array args (those alloc-ed with
           clCreateBuffer) *)
       if Generator_utilities.is_array_type formal.v_type then
         Environment.append env [Text("CALL_CL_GUARDED(");
                             Text("clReleaseMemObject, ");
                             Text(sprintf "(__arg%d)" arg_n);
                             Text(");\n")]
       else "", env
      in
      match formals with
       [] -> "", env
      | formal :: other_formals ->
        Environment.append env [Text("CALL_CL_GUARDED("); (* always free output arr *)
                            Text("clReleaseMemObject, ");
                            Text("(__arg1)");
                            Text(");\n");
                             Generator(generate_cl_release arg_n formal);
                            Generator(generate_cl_releases (arg_n + 1) other_formals)]
    in                     (* user args start at 2 *)
    Environment.append env [Generator(generate_cl_releases 2 fdecl.formals)]
  in
  let __arr_len = {
      v_type = "int";
      v_name = "__arr_len";
      a_size = -1;
    }
  in
  Environment.append env [Text(sprintf "%s %s("
                               (Generator_utilities.c_type_from_str fdecl.r_type)
                               fdecl.fname);
```

```
                        Generator(generate_formals_vdecl_list (__arr_len ::
fdecl.formals));
                       Text(")\n{\n");
                       Generator(generate_cl_arg_list fdecl);
                       Generator(generate_cl_enqueue_write_buffer_list fdecl);
                       Generator(generate_cl_set_kernel_args fdecl);
                       Generator(generate_cl_enqueue_nd_range_kernel fdecl);
                       Generator(generate_cl_enqueue_read_buffer fdecl);
                       Generator(generate_cl_release_list fdecl);
                       Text("return __out;\n");
                       Text("}\n")]

(* ------------------------------------------------------------------ *)
(* CPU functions                                                      *)
(* ------------------------------------------------------------------ *)

let generate_func_formals_and_body stmt_list vdecl_list env =
    Environment.append env [Generator(generate_formals_vdecl_list vdecl_list);
                            Text("){\n");
                            Generator(process_stmt_list (stmt_list));
                            Text("}\n")]
let rec generate_cpu_funcs fdecls env =
  let generate_cpu_func fdecl env =
    match fdecl.isGfunc with
      false ->
      let main_checked_name = function
         "main" -> "snuggle"
       | other_name -> other_name
      in
      Environment.append env [Env(add_func
                               fdecl.fname (Generator_utilities.fdecl_to_func_info
fdecl));
                    Env(update_curr_func fdecl.fname);
                          Text(sprintf "%s %s("
                                     fdecl.r_type (main_checked_name fdecl.fname));
                          NewScope(generate_func_formals_and_body
                                   fdecl.body fdecl.formals)]

    | true ->
       Environment.append env [Env(add_gfunc fdecl);
                           Env(add_func fdecl.fname
                                      (Generator_utilities.fdecl_to_func_info fdecl));
                           NewScope(generate_kernel_invocation_function fdecl)]
  in
  match fdecls with
    [] -> "", env
  | [fdecl] -> generate_cpu_func fdecl env
  | fdecl :: other_fdecls ->
     Environment.append env [Generator(generate_cpu_func fdecl);
                         Generator(generate_cpu_funcs other_fdecls);]

(* ------------------------------------------------------------------ *)
(* OpenCL Kernels                                                     *)
(* ------------------------------------------------------------------ *)
(* Each gfunc has a requires a set of variables to access its
   associated OpenCL kernel representation. For a gfunc called
   "mygfunc", these variables are:
   - `mygfunc_kernel_string'  : a string of the opencl kernel code
   - `mygfunc_kernel_name'    : a string of the function name;
                                circuitously, this will be "mygfunc"
   - `mygfunc_compiled_kernel : the compiled cl_kernel object

   We have to declare all of these variable globally (and at the top
   of our generated c program) so they will be accessible from any
   cpu function.

   ASSUMPTIONS:
```

```
   - the incoming func_info struct is typechecked
   - the first argument of the __kernel is the size for the whole function
   - the second argument of the __kernel is the output array *)

(* add the blocksize variables and then process the statment list
   *)
let rec generate_cl_kernel_body stmt_list fdecl env =
  Environment.append env
                   [Env(update_scope_add_var "__block_start" Int);
                    Env(update_scope_add_var "__block_end" Int);
                    Env(update_scope_add_var "_id" Int);
                    Text("const int _id = get_global_id(0);");
                    Text(sprintf "const int __block_start = _id * %d;"
                               fdecl.blocksize);
                    Text(sprintf "const int __block_end = _id * %d + %d;"
                               fdecl.blocksize
                               fdecl.blocksize);
                    Generator(process_stmt_list stmt_list);
                   ]

(* return a comma separated list of kernel formal declarations and
   adds the variables to the current scope *)
let rec generate_cl_kernel_vdecl_list vdecl_list env =
  let generate_cl_kernel_vdecl vdecl env =
    let c_type = Generator_utilities.c_type_from_str vdecl.v_type in
    let sheets_type = Generator_utilities.vdecl_type vdecl in
    Environment.append env [Env(update_scope_add_var vdecl.v_name sheets_type);
                         Text(sprintf "__global const %s %s" c_type vdecl.v_name)]
  in
  match vdecl_list with
    [] -> "", env
  | [vdecl] -> generate_cl_kernel_vdecl vdecl env
  | vdecl :: other_vdecls ->
     Environment.append env [Generator(generate_cl_kernel_vdecl vdecl);
                          Text(", ");
                          Generator(generate_cl_kernel_vdecl_list other_vdecls)]

let gfunc_to_cl_kernel_string gfdecl env =
  (* we have to reject all references to variables that aren't
     immediately in scope *)
  (* we're going to have to escape double-quotes when we write
     these string literals *)
  (* here we use KernelText instead of Text, which surrounds the
     string it takes in with quote marks and adds a newling at the end
     so that the text shows up in the generated c code as a multi-line
     string literal *)
  let base_r_type = Generator_utilities.c_type_from_str gfdecl.r_type in
  let sheets_r_type = Generator_utilities.str_to_type gfdecl.r_type in
  Environment.append env [
                     (* we have to manually modify scope because we're processing
                       gfunc bodies separately from their declarations*)
                     Env(update_curr_func gfdecl.fname);
                     Text("\"");
                       Env(update_on_gpu true);
                     Env(update_scope_add_var "__arr_len" Int);
                     Env(update_scope_add_var "__out" sheets_r_type);
                     Text("__kernel ");
                     Text(sprintf "void %s(__global const int __arr_len, __global
%s__out,"
                                gfdecl.fname base_r_type);
                     Generator(generate_cl_kernel_vdecl_list gfdecl.formals);
                     Text(")");
                     Text("{");
                     Generator(generate_cl_kernel_body gfdecl.body gfdecl);
                     Text("}");
                     Env(update_on_gpu false);
                     Text("\"");
```

```
                    ]

let gfunc_to_cl_kernel gfdecl env =
  Environment.append env [Text(sprintf "const char *%s_kernel_string =\n"
gfdecl.fname);
                       (* we aren't ever changing the environment
                       above the gfunc's scope, but we need to
                       generate a new scope to parse the gfunc's
                       contents *)
                       NewScope(gfunc_to_cl_kernel_string gfdecl);
                       Text(";\n");
                       Text(sprintf "const char *%s_kernel_name = \"%s\";\n"
                                   gfdecl.fname gfdecl.fname);
                       Text(sprintf "cl_kernel %s_compiled_kernel;\n" gfdecl.fname)]

let rec gfunc_list_to_cl_kernels gfdecl_list env =
  match gfdecl_list with
    [] -> "", env
  | gfdecl :: other_gfdecls ->
     Environment.append env [Generator(gfunc_to_cl_kernel gfdecl);
                          Generator(gfunc_list_to_cl_kernels other_gfdecls)]

let generate_cl_kernels env =
  let cl_globals = "cl_context __sheets_context;\n"
                 ^ "cl_command_queue __sheets_queue;\n"
                 ^ "cl_int __cl_err;\n"
  in
  Environment.append env [Text(cl_globals);
                       Generator(gfunc_list_to_cl_kernels env.gfunc_list)]

(* ---------------------------------------------------------------- *)
(* Main: opencl context creation and frees                           *)
(* ---------------------------------------------------------------- *)

let rec generate_compile_kernels gfdecl_list env =
  let generate_compile_kernel gfdecl =
    sprintf "%s_compiled_kernel = kernel_from_string(__sheets_context,
%s_kernel_string, %s_kernel_name, SHEETS_KERNEL_COMPILE_OPTS);\n" gfdecl.fname
gfdecl.fname gfdecl.fname
  in
  match gfdecl_list with
    [] -> "", env
  | gfdecl :: other_gfdecls ->
     Environment.append env [Text(generate_compile_kernel gfdecl);
                          Generator(generate_compile_kernels other_gfdecls)]

let rec generate_release_kernels gfdecl_list env =
  let generate_release_kernel gfdecl =
    sprintf "CALL_CL_GUARDED(clReleaseKernel, (%s_compiled_kernel));\n" gfdecl.fname
  in
  match gfdecl_list with
  [] -> "", env
  | gfdecl :: other_gfdecls ->
     Environment.append env [Text(generate_release_kernel gfdecl);
                          Generator(generate_release_kernels other_gfdecls)]

let generate_main env =
  Environment.append env [Text("int main()\n");
                       Text("{\n");
                       Text("create_context_on(SHEETS_PLAT_NAME, SHEETS_DEV_NAME, 0,
&__sheets_context, &__sheets_queue, 0);\n");
                       Generator(generate_compile_kernels env.gfunc_list);
                       Text("snuggle();\n");
                       Generator(generate_release_kernels env.gfunc_list);
                       Text("CALL_CL_GUARDED(clReleaseCommandQueue,
(__sheets_queue));\n");
                       Text("CALL_CL_GUARDED(clReleaseContext,
```

```
(__sheets_context));\n");
                      Text("return 0;\n");
                      Text("}\n")]

(* ------------------------------------------------------------------ *)
(* Parse and print                                                    *)
(* ------------------------------------------------------------------ *)

let _ =
  let lexbuf = Lexing.from_channel stdin in
  let vdecls, fdecls = try
      Parser.program Scanner.token lexbuf
    with except ->
      let curr = lexbuf.Lexing.lex_curr_p in
      let line = curr.Lexing.pos_lnum in
      let col = curr.Lexing.pos_cnum in
      let tok = Lexing.lexeme lexbuf in
      raise (SyntaxError (line, col, tok))
  in
  let env = Environment.create in
  let global_vdecls, env = generate_global_vdecl_list (List.rev vdecls) env in
  let cpu_funcs, env = generate_cpu_funcs (List.rev fdecls) env in
  let cl_kernels, env = generate_cl_kernels env in
  let main, _ = generate_main env in
  print_string ("#include <stdio.h>\n"
              ^ "#include \"aws-g2.2xlarge.h\"\n"
              ^ "#include \"cl-helper.h\"\n"
              ^ "#include \"timing.h\"\n"
              ^ "#include <CL/cl.h>\n\n"
                ^ "#define time_start() get_timestamp(&start)\n"
                ^ "#define time_end() get_timestamp(&end)\n"
                ^ "timestamp_type start;\n"
                ^ "timestamp_type end;\n");
  print_string cl_kernels;
  print_string global_vdecls;
  print_string cpu_funcs;
  print_string main
```

8.6. Generator Utilities

```
(*
 * Sheets Code Generator Utilities
 *
 * Authors: Ruchir Khaitan, Ben Barg, Amelia Brunner
 * Copyright 2014, Symposium Software
 *)

open Ast;;
open Environment;;
open Printf;;

exception TypeError of string;;
exception NotImplementedError of string;;
exception UnsupportedArrayTypeError;;

let eval_basic_binop type1 type2 =
    if (type1 = type2) then
        match type1 with
            | Int -> type1
            | Float -> type1
            | Array (Int) -> type1
            | Array (Float) -> type2
            | _-> raise (TypeError("Types not ID, int, or float"))
```

```
    else
        raise (TypeError("Incompatible types"))

let eval_binop type1 type2 op =
    match op with
    | Plus -> (eval_basic_binop type1 type2)
    | Minus -> (eval_basic_binop type1 type2)
    | Times -> (eval_basic_binop type1 type2)
    | Divide -> (eval_basic_binop type1 type2)
    | Equal -> (eval_basic_binop type1 type2)
    | Greater -> (eval_basic_binop type1 type2)
    | Less -> (eval_basic_binop type1 type2)
    | Geq  -> (eval_basic_binop type1 type2)
    | Leq  -> (eval_basic_binop type1 type2)
    | Neq    -> (eval_basic_binop type1 type2)
    | _->      raise (TypeError("Incompatible types"))

let eval_array_acc array_ int_expr =
    match array_ with
        | Array ( Int) ->
                (match int_expr with
                | Int -> Int
                | _-> raise (TypeError("Cannot access element in array with non-int
datatype")))
        | Array ( Float) ->
                (match int_expr with
                | Int -> Float
                | _-> raise (TypeError("Cannot access element in array with non-int
datatype")))
        | _-> raise (TypeError("Cannot access element in non-array type"))

(* Returns typeof block access *)
let typeof_block_acc id env =
    match id with
   "start" -> Int
   | "end" -> Int
   | "out" -> (Environment.return_typeof_func env.current_function env)
   | _-> raise (TypeError("Invalid block access"))
let rec expr_typeof expr env =
  match expr with
    Literal_int(i) -> Int
  | Literal_float(f) -> Float
  | Literal_int_a(i_a) -> Array( Int )
  | Literal_float_a(i_a) -> Array(Float)
  | Id(s) -> Environment.typeof s env
  | Binop(exp1, op, exp2) -> (eval_binop (expr_typeof exp1 env) (expr_typeof exp2 env)
op)
  | ArrayAcc(exp1, exp2) -> (eval_array_acc (expr_typeof exp1 env) (expr_typeof exp2
env) )
  | Call(func_id, expr_list ) -> (typeof_func_call func_id expr_list
(Environment.get_func_args func_id env) env)
  | BlockAcc(id, _) -> (typeof_block_acc id env)
  | _-> raise (NotImplementedError("Undefined type of expression"))

and typeof_func_call func_id expr_list arg_list env =
  (* First make sure that all of the arguments are valid, then check
     all ids, then return the type of the function *)
  let rec check_expr_list expr_list arg_list =
    match expr_list, arg_list with
    | [],[] -> Environment.return_typeof_func func_id env
    | expr1::other_exprs, arg1::other_args ->
       if((expr_typeof expr1 env) != arg1 ) then
         raise (TypeError("Function arguments of incorrect type"))
       else
         check_expr_list other_exprs other_args
    | _-> raise (TypeError("Incorrect number of function arguments"))
  in
```

```
  match func_id with
    "printf" -> Int
  | _ -> check_expr_list expr_list arg_list

let str_to_type str =
        match str with
        "int" -> Int
       | "float" -> Float
       | "int[]" -> Array(Int)
       | "float[]" -> Array(Float)
       |  _-> raise (NotImplementedError("Unrecognized type " ^ str))

let rec typecheck_stmt stmt env = true

let rec typecheck_stmt_list stmt_list env =
  match stmt_list with
    [] -> true
  | stmt :: rest_of_stmts -> typecheck_stmt stmt env;
                             typecheck_stmt_list rest_of_stmts env
let vdecl_type vdecl =
  str_to_type vdecl.v_type

let rec vdecl_list_to_type_list vdecl_list =
  match vdecl_list with
    vdecl::rest_of_vdecls -> (vdecl_type vdecl)::(vdecl_list_to_type_list
rest_of_vdecls )
  | [] -> []

let rec vdecl_list_to_string_list vdecl_list =
    match vdecl_list with
        vdecl::rest_of_vdecls -> (vdecl.v_name)::(vdecl_list_to_string_list
rest_of_vdecls )
       | [] -> []

let fdecl_to_func_info fdecl =
  {
    id = fdecl.fname;
    gpu = fdecl.isGfunc;
    return = str_to_type fdecl.r_type;
    args = vdecl_list_to_type_list fdecl.formals;
    arg_names = vdecl_list_to_string_list fdecl.formals;
    _blocksize = fdecl.blocksize;
  }

let arr_type_str_to_base_type = function
    "float[]" -> "double"
  | "int[]" -> "int"
  | "float[][]" -> "double"
  | "int[][]" -> "int"
  | _ -> raise UnsupportedArrayTypeError

let c_type_from_str = function
    "int" -> "int"
  | "float" -> "double"
  | "float[]" -> "double *"
  | "int[]" -> "int *"
  | "float[][]" -> "double **"
  | "int[][]" -> "int **"
  | _ -> raise UnsupportedArrayTypeError

let is_array_type = function
    "int" | "float" -> false
    | _ -> true
```

8.7. Environment Types

```
(*
 * Sheets Environment Types
 *
 * Authors: Ruchir Khaitan, Ben Barg, Amelia Brunner
 * Copyright 2014, Symposium Software
 *)

open Ast;;

module VariableMap = Map.Make(String);;
module FunctionMap = Map.Make(String);;

exception EmptyEnvironmentError;;
exception NameAlreadyBoundError of string;;
exception VariableNotFound of string;;
exception VariableAlreadyDeclared;;
exception AlreadyDeclaredError;;
exception FunctionNotDefinedError;;
exception ReservedWordError of string;;

type func_info  = {
       id : string;
        gpu : bool;
       return : datatype;
       args : datatype list;
       arg_names: string list;
       _blocksize : int;
}

(* Record indicating what the current environment keeps track of *)
type env = {
    var_stack: datatype VariableMap.t list;
    func_return_type_map: func_info FunctionMap.t;
    current_function: string;
    on_gpu: bool;
    gfunc_list: fdecl list; (* we need to save the body of the
                                   gfunc so we can do kernel string
                                   generation *)
    num_array_returns : int;
    var_array_sizes : int VariableMap.t;
  }

(* Types that can be returned by the generator as it modifies either
   the text of the generated code changes the environment as it is
   parsing the file or passes fuunctions to edit both code and
   environment either in the existing scope or in a new scope *)
type source =
  | Text of string
  | Env of (env -> env)
  | Generator of (env -> (string * env))
  | NewScope of (env -> (string * env))

(* Create initializes an empty record for environment *)
let add_default_func f_id fmap =
  let f_info =
    {
      id = f_id;
      gpu = false;
      return = Int;
      args = [];
      arg_names = [];
      _blocksize = -1;
    }
```

```
  in
  (FunctionMap.add f_id f_info fmap)

let init_func_map =
  let time_start_fmap =
    let printf_fmap =
      add_default_func "printf" FunctionMap.empty
    in
    add_default_func "time_start" printf_fmap
  in
  add_default_func "time_end" time_start_fmap

let create =
  {
    var_stack = VariableMap.empty::[];
    func_return_type_map = init_func_map;
    current_function = ""; (* TODO maybe this needs a better convention *)
    on_gpu = false;
    gfunc_list = [];
    num_array_returns = 0;
    var_array_sizes = VariableMap.empty;
  }

(* Update gives a new env record with updated values of the record *)
let update v_stack f_map curr_f gpu g_list num_arr_ret  v_a_s=
  {
    var_stack = v_stack;
    func_return_type_map = f_map;
    current_function = curr_f;
    on_gpu = gpu;
    gfunc_list = g_list;
    num_array_returns = num_arr_ret;
    var_array_sizes = v_a_s;
  }

(* Functions that let us modify only one variable in environment at a
   time *)
let update_only_scope new_scope env =
  update new_scope env.func_return_type_map env.current_function env.on_gpu
env.gfunc_list env.num_array_returns env.var_array_sizes

let update_only_func new_func env =
  update env.var_stack new_func env.current_function env.on_gpu env.gfunc_list
env.num_array_returns env.var_array_sizes

let update_curr_func new_curr_func env =
  update env.var_stack env.func_return_type_map new_curr_func env.on_gpu env.gfunc_list
env.num_array_returns env.var_array_sizes

let update_on_gpu gpu env =
  update env.var_stack env.func_return_type_map env.current_function gpu env.gfunc_list
env.num_array_returns env.var_array_sizes

let update_gfunc_list g_list env =
  update env.var_stack env.func_return_type_map env.current_function env.on_gpu g_list
env.num_array_returns env.var_array_sizes

let increment_num_array_rets env =
  update env.var_stack env.func_return_type_map env.current_function env.on_gpu
env.gfunc_list (env.num_array_returns + 1) env.var_array_sizes

let add_array_size id size env =
  update env.var_stack env.func_return_type_map env.current_function env.on_gpu
env.gfunc_list env.num_array_returns  (VariableMap.add id size env.var_array_sizes)

(* Checks all scopes to see if variable has been declared *)
let is_var_in_scope id env =
```

```
  let rec check_scopes scope_stack =
    let check_level scope_level =
      VariableMap.mem id scope_level
    in
    match scope_stack with
    | [] -> false
    | [scope_level] ->
       if env.on_gpu then false
       else check_level scope_level
    | scope_level :: other_scope_levels ->
       if check_level scope_level then true
       else check_scopes other_scope_levels
  in check_scopes env.var_stack

(* Checks all scopes to find variable and returns type if found
 * Raises exception if not found *)
let typeof id env =
  let rec check_scopes scope_stack =
    match scope_stack with
    | [] -> raise (VariableNotFound id)
    | scope_level :: other_scope_levels ->
       if VariableMap.mem id scope_level then
         VariableMap.find id scope_level
       else
         check_scopes other_scope_levels
  in check_scopes env.var_stack

(* Adds a variable to the topmost level of the variable stack. Does
   not check if variable is already in stack do elseware raises error
   if stack is an empty list returns an updated environment with an
   updated scope stack *)
let update_scope_add_var id datatype env =
  let old_scope, scope_tail =
    ( match env.var_stack with
      | old_scope :: scope_tail -> old_scope, scope_tail
      | [] -> raise EmptyEnvironmentError ) in
  let new_scope = VariableMap.add id datatype old_scope in
  update_only_scope (new_scope::scope_tail) env

(* Handles adding a variable to the current environment.

   Takes a tuple of
   - id : string
   - datatype
   - (text: string, env : env)

   and either raises
   - EmptyError
   - AlreadyDeclaredError

   or it adds the variable to the current top scope with
   update_scope_add_var and returns a updated env *)
let add_var id datatype env =
    match env.var_stack with
      | [] -> raise EmptyEnvironmentError
      | scope_level :: scope_tail ->
              if VariableMap.mem id scope_level then
                      raise VariableAlreadyDeclared
              else
              update_scope_add_var id datatype env

let get_arr_size id env =
    if VariableMap.mem id env.var_array_sizes then
      (VariableMap.find id env.var_array_sizes)
    else
     raise VariableAlreadyDeclared
```

```
(* adds a new empty variableMap to the top of var stack used to enter
   a subscope takes an env, returns an updated env *)
let push_scope env =
    update_only_scope (VariableMap.empty::env.var_stack) env

(* removes a variableMap from the top of var stack used to enter a
   subscope takes an env, returns an updated env *)
let pop_scope env =
   match env.var_stack with
    | popped_scope::other_scopes ->
        update_only_scope other_scopes env
    | [] -> raise EmptyEnvironmentError

(* ------------------------------------------------------- *)
(* The following methods deal with handling the function map *)
(* ------------------------------------------------------- *)

(* Takes a function id and checks if the function is defined
   returns bool *)
let is_func_declared id env =
   FunctionMap.mem id env.func_return_type_map

(* Takes a function id and either returns the mapped function info
   or raises an undefined function error *)
let get_func_info id env =
  if is_func_declared id env then
    FunctionMap.find id env.func_return_type_map
  else
    raise FunctionNotDefinedError

let get_func_args id env =
    (get_func_info id env).args

(* Inserts a new function to the function map and updates environment
   or raises a AlreadyDeclaredError *)
let add_func id finfo env =
  if is_func_declared id env then
    raise VariableAlreadyDeclared
  else
    update_only_func (FunctionMap.add id finfo env.func_return_type_map) env

(* Returns the datatype of a function or
   raises a undefined error if the function is not defined *)
let return_typeof_func id env =
    let f_info =
       get_func_info id env in
    f_info.return

let id_is_gfunc id env =
  let rec name_in_gfunc_list gfunc_list =
    match gfunc_list with
      [] -> false
    | gfunc :: other_gfuncs ->
       if id = gfunc.fname then true
       else name_in_gfunc_list other_gfuncs
  in
  name_in_gfunc_list env.gfunc_list

let rec check_gfunc_name_in_list glist gfunc_fdecl =
  match glist with
    [] -> false
  | gfunc_fdecl :: rest_of_gfuncs -> if gfunc_fdecl.fname = gfunc_fdecl.fname then true
                                     else (check_gfunc_name_in_list rest_of_gfuncs
gfunc_fdecl)

let is_gfunc_declared gfunc_fdecl env =
  if check_gfunc_name_in_list env.gfunc_list gfunc_fdecl then
```

```
    raise (AlreadyDeclaredError)
  else if is_func_declared gfunc_fdecl.fname env then
    raise (AlreadyDeclaredError)
  else false

let add_gfunc gfunc_fdecl env =
  if (is_gfunc_declared gfunc_fdecl env) then
    raise (AlreadyDeclaredError)
  else if (gfunc_fdecl.fname = "main") then
    raise (ReservedWordError("a gfunc cannot be the main method"))
  else
    update_gfunc_list (gfunc_fdecl :: env.gfunc_list) env

let quote_and_strip_newline str =
  let len = String.length str in
  let last_char s = s.[(len - 1)] in
  if str = "" then ""
  else
    match last_char str with
      '\n' -> (String.sub str 0 (len - 2))
    | _    -> str

(* This is the running loop of the codegen step. First, f is a
   function that takes a (text, environment) tuple and matches it with
   a component which is either

   * some new text - gets appended to existing text
   * some modified environment - replaces existing environment
   * some function named gen applied to the same scope

   gen is applied to the existing environment and gen returns a tuple
   of (string, env). The text gets appended to the existing text, and
   the new env replaces the old env.

   Note that gen can be a function also with arguments given to it
   then this function f is applied to component in the list of
   components always returning an updated (text, env) tuple that is
   passed to the next component in the list *)
let append init_env components =
  let f (text, env) component =
    match component with
    | Text(str) -> if env.on_gpu then
                     text ^ (quote_and_strip_newline str), env
                   else
                     text ^ str, env
    | Env (env_gen) -> let new_env = env_gen env in
                       text, new_env
    | Generator(gen) -> let new_str, new_env = gen env in
                       text ^ new_str, new_env
    | NewScope(gen) -> let new_str, new_env = gen (push_scope env) in
                       text ^ new_str, pop_scope new_env in
  List.fold_left f("", init_env) components
;;
```

8.8. Testing Script

```
#!/bin/bash

# Sheets Automated Testing Script
# Author: Gabriel Blanco
# Copyright 2014, Symposium Software
```

```
executable="./test_env.sh"
output_file="envtests.output"

rm -f *.proc.sht
rm -f envtests.output
environment_tests=$(find . -name "*\.sht")

path_to_name()
{
    local fullpath=$1
    test_path="${fullpath%.*}"   # Strip Extension
    test_name="${test_path##*/}" # Strip Preceding Path
}

check_if_fail()
{
    local name=$1
    shouldfail=false
    if [[ "$name" == *"_n_"* ]] ; then
        shouldfail=true
    fi
}

echo "Running all environment tests in current directory"
echo ""
echo "  Note: an 'x' next to a test indicates that the test"
echo "  is failing when it is expected to pass, or passing"
echo "  when expected to fail. This does not guarantee that"
echo "  a successful test is passing or failing for the"
echo "  expected reason, so make sure to verify that all"
echo "  outputs are correct"
echo ""

echo "Environment Testing Suite" >> $output_file

for file in $environment_tests ; do
    path_to_name $file
    check_if_fail $test_name

    echo "" >> $output_file
    echo "===================================" >> $output_file

    VAR=$( ./$executable $test_name 2>&1 )
    echo "$VAR" >> $output_file

    fatal_error=$(echo $VAR | grep 'Fatal error:')

    if [[ $shouldfail == true ]] ; then
        if [[ $fatal_error == '' ]] ; then
            echo "[x] Running Test: $test_name"
        else
            echo "[ ] Running Test: $test_name"
        fi
    else
        if [[ $fatal_error == '' ]] ; then
            echo "[ ] Running Test: $test_name"
        else
            echo "[x] Running Test: $test_name"
        fi
    fi

    echo "===================================" >> $output_file
    echo "" >> $output_file
done

echo "done"
```

```
echo "OUTPUT FILE: '$output_file'"

rm -f *.proc.sht

exit
```