Flow

A Programming Language for Kahn Process Networks

github.com/mgouzenko/flow-lang

Programming Languages and Translators
COMS W4115
Fall 2015

Adam Chelminski (apc2142)
Zachary Gleicher (zjg2012)
Mitchell Gouzenko (mag2272)
Hyonjee Joo (hj2339)

Table of Contents

1 Introduction

Flow is a programming language based on the model of Kahn Process Networks (KPNs). KPNs are a model of distributed computation characterized by parallel processes interconnected with one-way FIFO communication channels. The rules governing KPNs cause them to execute in a deterministic fashion, regardless of independent process scheduling. KPNs have applications in distributed systems, signal processing, and statistical modeling where streams of data need to be transformed and parallel processing is advantageous. The goal of Flow is to enable users to programmatically generate KPNs.

2 Language Tutorial

Flow makes use of two fundamental abstractions: processes and channels. Processes are nodes in the KPN. They perform work, and communicate with each other via channels. Channels are FIFO structures that hold a series of tokens. Processes can pass tokens around by reading from and writing to channels. A channel may have no more than one reading and writing process, respectively. Processes may not check if a channel is empty, as this would ruin determinism. Channels are the one and only means by which processes communicate.

Flow uses syntax that is generally similar to C, with added constructs for channels and processes. The following example program shows how to declare, define, and connect processes to find the running sum of a sequence of integers. In this section, we will examine the program piece-by-piece.

```
// numGen process generates stream of ints
proc num_gen(out int ochan){
  list <int> nums = [1, 2, 3, 4, 5];
  while(#nums > 0) {
    @nums -> ochan;
    nums = ^nums;
  }
  poison ochan;
}

// sum process outputs running total for input stream
proc sum(in int chan) {
  int sum = 0;
  while(chan) {
    sum = sum + @chan;
    print_int(sum);
  }
}

// main contains channel declarations & connects and launches processes
int main() {
  channel<int> chan;
  num_gen(chan);
  sum(chan);
}
```

2.1 num_gen

`num_gen` is a process which generates and outputs a stream of integers. It is declared almost like a c-style function, but it uses the `proc` keyword, indicating that it's a process. It accepts a parameter called "`ochan`", which is declared as "`out int ochan`". This signifies that `ochan` is a channel of ints. Furthermore, `ochan` is an `out` channel, indicating that `num_gen` can only write tokens to it.

```
// numGen process generates stream of ints
proc num_gen(out int ochan){
  list <int> nums = [1, 2, 3, 4, 5];
  while(#nums > 0){
    @nums -> ochan;
    nums = ^nums;
  }
  poison ochan;
}
```

Looking at the process body, we see that there is an list of integers called `nums`, initialized to the numbers 1 through 5. The while loop tests if `#nums > 0`. The unary operator `#` returns the length of the list.

Inside the body of the while loop, the contents of `nums` are written to ochan in the statement `@nums -> ochan`. In this statement, `@nums` gets the head of the list (which would be the integer 1 on the first pass). The `->` operator sends the head of the list to `ochan`. Finally, the statement `nums = ^nums` sets nums equal to its tail, effectively iterating through the list.

At the end of the process, we send a poison token to the output channel with the `poison` keyword. This is to signal to the process connected to `ochan's` read end that `num_gen` will no longer write to `ochan`. After poisoning `ochan`, `num_gen` terminates. Note that if a process terminates without poisoning channels that it writes to, those channels will be poisoned automatically.

2.2 sum

```
// sum process outputs running total for input stream
proc sum(in int chan) {
  int sum = 0;
  while(chan) {
    sum = sum + @chan;
    print_int(sum);
  }
}
```

The `sum` process calculates and prints to stdout a running sum of the integers it receives. Like `num_gen`, `sum` takes an int channel argument. This argument - called `chan` - is declared with the keyword `in`, meaning that that `sum` can only read from it.

Before reading from a channel, we need to check whether there are tokens to read. Channel used as booleans within conditional clauses (i.e. `while(chan)`) block until there is a token to be read from the channel. When there is a token to be read, the channel reference returns true. If the channel is empty and has been poisoned (indicating that no additional tokens will be sent to the channel), the channel reference returns false.

To retrieve the next token from an input channel, we use the `@` operator. In the sum process, we add the integer retrieved from the channel to a sum total and print out the current sum after each integer observed. `print_int()` is a built-in function in Flow.

2.3 main

```
// main contains channel declarations & connects and launches processes
int main() {
  channel<int> chan;
  num_gen(chan);
  sum(chan);
}
```

The main method is where we declare the integer channel used to connect the `num_gen` and `sum` processes. Invoking the `num_gen` and `sum` processes in the main method launches them on separate threads. The main method will end when all processes terminate.

2.3 Running the program

To run the program, use the script `run.sh` with the name of the program file as the command line argument. If you would like to generate the c code of a flow program, run `make` and then run `./flowc` with the name of the program file as the command line argument.

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Identifiers

An identifier is a name, consisting of ASCII letters, digits, and '_' characters. The first character of the identifier must be a letter. Identifiers are case-sensitive. Below is the parsing rule for an identifier:

```
IDENTIFIER :=
    ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

3.1.2 Key Words

Keywords are reserved; they have syntactic and semantic purposes and thus cannot be used as identifiers. The keywords in Flow are:

```
int
double
char
bool
string
void
list
proc
channel
false
true
```

```
if
else
for
while
continue
break
return
poison
in
out
```

3.1.3 Comments

The characters `'//'` introduces a new comment. The rest of the line after `'//'` will be part of a comment.

3.1.4 Punctuation

The punctuators of our language are listed below along with examples.

Punctuator	Use	Example
`,`	list separator	`sample_list = [0, 1, 2];`
`[]`	list delimiter	`sample_list = [1];`
`()`	conditional delimiter, function call, expression grouping	`while (bool)`
`{}`	statement block	`if (cond) { statements }`
`;`	statement end	`x = 0;`
`‘`	char literal delimiter	`c = ‘a’;`
`"`	string literal delimiter	`x = "hello";`

3.1.5 Operators

Operator	Use	Associativity
`@`	Retrieve token from channel or value at head of list	Non-associative

*	Multiplication	Left
/	Division	Left
%	Modulo	Left
+	Addition	Left
-	Subtraction	Left
=	Assignment	Right
==	Equal to	Left
!=	Not equal to	Left
<	Less than	Left
>	Greater Than	Left
<=	Less than or equal to	Left
>=	Greater than or equal to	Left
&&	short circuit logical AND	Left
||	short circuit logical OR	Left
!	negation	Non-associative
->	Send item to channel	Left
::	Concatenation to front of list	Right
#	Get list length	Non-associative
^	Return list tail	Non-associative

The operators are listed from greatest to least precedence:

1. @
2. ! ^ #
3. * / %
4. + -
5. ::
6. < > <= >=
7. == !=
8. && ||
9. =
10. ->

3.1.6 Whitespace

White spaces include blanks, tabs, and newline characters. Flow is not whitespace sensitive. Blocks of code are delimited by curly braces, not indentation.

3.2 Types

3.2.1 Primitive Types

3.2.1.1 Integer Type

An integer is a signed 4 byte sequence of digits. An integer literal is a sequence of digits preceded by an optional negative sign. A single zero cannot be preceded by a negative sign.

```
int x = 0;
int y = -1;
int z = 100;
```

3.2.1.2 Double Type

A double type is a signed 8 byte double-precision floating point data type consisting. A double literal contains an optionally signed integer part, a decimal point and a fractional part. Either the integer part or the fractional part can be missing, but not both.

```
double a = 0.1;
double b = -1.1;
double i = 1.;
double j = .2;
```

3.2.1.3 Boolean Type

A boolean literal is either the `true` keyword or the `false` keyword, and occupies 1 byte. A boolean is its own type and cannot be compared to a non-boolean variable. Therefore, evaluating false == 0, would result in an error.

```
bool x = true;
bool y = false;
```

3.2.1.4 Char Type

A char literal is a single character surrounded by single quotes.

```
char x = 'a';
```

3.2.1.5 Void Type

The `void` type can be used to declare a function that does not return anything. It has no other use in the Flow language.

3.2.2 Non-Primitive Types

In Flow there are 5 non-primitive types: strings, lists, channels, and processes.

3.2.2.1 String Type

A string is a sequence of characters. A string literal is placed between double quotes. String literals are sequences of ASCII characters, enclosed by double quotes. Strings are immutable. Declared strings are automatically initialized to the empty string `""`;

```
string name = "Steven";
```

Strings support the following built-in print function:

- `print_string(string a)`
    - Prints the given string, no newline appended.

3.2.2.2 List Type

A list is immutable and contains elements of the same type. Supported list types include: integers, characters, doubles, and directionless channels.

```
list <int> test_list = [1, 2, 3];
list <char> empty_list = [];
list <double> new_list = old_double_list;
```

List concatenation can be done with the `::` operator. The binary operator takes a list element as a left operand and a list as a right operand and returns the new list head. List concatenation only allows for elements to be added to the front of the list.

```
list <char> test_list = ['h', 'a', 't'];
test_list = 'c'::test_list;  // test_list is now ['c', 'h', 'a', 't']

list <int> growing_list;
growing_list = 0::growing_list;
growing_list = 1::growing_list;
growing_list = 2::growing_list;    // growing_list is now [2, 1, 0]
```

The unary operators for lists are `#`, `@`, and `^`. `#` returns the length of the list. `@` returns the element at the head of the list without modifying the list. `^` returns the tail of the list.

```
int length = #test_list;          // length = 4
char first_char = @test_list;     // first_char = 'c'
test_list = ^test_list;           // test_list is now ['h', 'a', 't']
```

3.2.2.4 Process Type

In Flow, a process is an independent unit that performs work on tokens from zero or more channels. The process type allows the programmer to define the work done at a node in the Kahn Process network.

A process may act as a sender for zero or more channels, as well as a receiver for zero or more channels. The workflow for deploying a process consists of first defining the process and then invoking it with the necessary arguments. In a compiled Flow program, each process runs on a separate thread.

Process declarations are further discussed in section 3.4.

3.2.2.5 Channel Type

Channels are FIFO structures that connect processes to other processes. At any time, a channel may contain a buffer of zero or more tokens - elements that have been been sent to that channel but not removed from it. The tokens that a channel holds must be of a uniform type that is determined by its declaration.

Channel declarations are of the following form:

```
channel<channel_type> IDENTIFIER;
```

```
// int_channel is a channel for integer tokens
channel<int> int_channel;
```

A channel must be bound to exactly one sending process and one receiving process. Only the bound sending process may send tokens to the channel, and only the bound receiving process may receive tokens from the channel. The receiving process is guaranteed to receive tokens in the order in which they were sent.

Channels may not be queried for size, nor can the next item in a channel be read without removing it from the channel. Once a channel is bound, it cannot be unbound.

Section 3.5 discusses specifics on how to use channels.

3.3 Functions

Functions can only be declared at the top level. Function declarations are as follows:

```
return_type IDENTIFIER( arg_declaration_list ) { stmt_list }
```

Function arguments may be primitive flow types, strings, lists, and channels. The argument declaration list is an optional list of comma separated expressions of the form: `arg_type arg_name` for all argument types except for channels. Refer to section 3.5.1 for details on passing channels as arguments.

The return type of a function must be a primitive type, channel, or list. The keyword `return` can be used in the function's body to return control of the program to the calling function or process. The type of the expression following the `return` keyword must match the return type in the function's declaration.

Function calls can be made with the the () punctuator and list of appropriate arguments.

```
// Function declaration and definition
int sum(int x, int y) {
       return x + y;
}

int i = sum(1, 2); # i == 3
```

3.4 Processes

Processes can be declared with a list of arguments but no return type. Like functions, processes may only be declared at the top level. A process is declared with the `proc` keyword as follows:

```
proc IDENTIFIER( arg_declaration_list ) { stmt_list }
```

Processes do not have a return type, but the return statement can be invoked in the process body to terminate the process. The return statement must not have a return value.

```
proc process_that_does_nothing(){
     return;   // this will terminate the process
     return 0; // this will throw an error at compile time
}
```

As an example, here is the definition of a process that interleaves two input streams and produces one output stream. This example uses the `@` and `->` operators and the `poison` keyword, which are discussed in Section 3.5.

```
proc interleaver(in int inchan1, in int inchan2, out int ochan){
       while(inchan1 || inchan2) {
          if(inchan1) {@inchan1 -> ochan;}
          if(inchan2) {@inchan2 -> ochan;}
     }
     poison ochan;
}
```

Processes are invoked with the the () punctuator and list of appropriate arguments. As soon as a process is invoked, it begins to run on a separate thread. Declaring the process defines the work it does, while invoking a process turns it into an actual node in the Kahn Process Network. Processes may be invoked from anywhere in the code, including from within the bodies of other processes.

```
int main() {
     channel<int> chan1;
     channel<int> chan2;
     channel<int> chan3;
     ....
     // invocation of the interleaver process
     interleaver(chan1, chan2, chan3);
}
```

3.5 Channels

3.5.1 Channel as Arguments

Channels may be passed as arguments to both processes and functions. Channels without a direction (eg. `channel<int> chan1`) can be passed into a function or process, but directionless channels cannot be read from or written to. Alternatively, channels with a direction can be passed into a process or function, following this pattern:

```
DIRECTION TYPE IDENTIFIER
```

where `DIRECTION` is either `in` or `out`, `TYPE` is the type of token the channel holds, and `IDENTIFIER` is the channel's identifier. If `DIRECTION` is `in`, the channel may only be used to read tokens. Conversely, if

`DIRECTION` is `out`, the channel may only be used to send tokens. To reiterate: this syntax is valid only in the argument declaration list for functions and processes.

In our `interleaver` example above (Section 3.4), the process takes three channels as arguments. The first two, `inchan1` and `inchan2`, are input channels declared with the `in` direction. The `interleaver` process can only read items from these channels, and never write to them. The last argument, `ochan`, is an output channel declared with the `out` direction. The `interleaver` can only write to this channel, and never read items from it.

3.5.2 Writing to Channels

Reading from and writing to channels is done with the `@` and `->` operators.

The `->` operator sends a token to a channel. This binary operator expects an expression as the left operand and an `out` channel identifier as the right operand. The expression is evaluated and the result (the token) is sent to the channel. The expression must evaluate to a value whose type matches the type of the channel being written to.

3.5.3 Poisoning Channels

Some processes may not terminate and the channels that the process writes to may be left open indefinitely until the program is forcefully terminated. However, in the case when a process terminates, it should send a `poison` token to the `out` channels bound to it. A poison token can be thought of as an EOF (end-of-file) for channels. This effectively closes the write end of the channels and indicates to processes that are waiting to read from these channels that no more tokens will be sent. Flow will automatically poison all output channels connected to a terminated process if the programmer does not explicitly do so.

3.5.2 Reading from Channels

The `@` operator is a unary operator on channel identifiers which returns the next token in the channel. Specifically, `@` can only operate on channels with direction `in`. If the channel is empty an error will be thrown, so it is the programmer's responsibility to check the status of the channel in a conditional statement before reading from it.

Checking the status of a channel can be done in a conditional statement. Consider the statement:

```
while(chan) {   // Chan is an arbitrary channel
        ...
}
```

When channels are evaluated in conditional statements like the one above, there are three scenarios.

1. **The channel is poisoned and empty**: the conditional will evaluate to false, because the channel will never possess another token.
2. **The channel is nonempty**: the conditional will evaluate to true, because there are tokens to be read from the channel.
3. **The channel is empty (but not poisoned)**: the evaluation of the conditional will block until either the channel is poisoned, or the channel is written to.

Let us revisit the interleaver process, introduced in Section 3.4:

```
proc interleaver(in int inchan1, in int inchan2, out int ochan){
       while(inchan1 || inchan2) {
          if(inchan1) {@inchan1 -> ochan;}
          if(inchan2) {@inchan2 -> ochan;}
     }
     poison ochan;
}
```

The expression `inchan1 || inchan2` in the while loop condition blocks if either of the input channels is empty but not poisoned. If both input channels are poisoned and empty, then the expression will evaluate to false and the while loop will terminate. But, if one channel is poisoned and empty, the statement will evaluate to false only if the other channel is poisoned and empty as well.

Once inside the while loop, we need to check the status of the input channels to determine which channel(s) can be read from. This is done by encapsulating the channel reads inside if statements. If `ichan1` can be read from, the expression `@inchan1 -> ochan` fetches a token from `inchan1`, and then sends it to `ochan`. If `inchan2` can be read from, the expression `@inchan2 -> ochan` fetches a token from `inchan2`, and then sends it to `ochan`. When the `interleaver` process finishes interleaving its inputs, it poisons the output channel with `poison ochan`. This is good programming practice in Flow. If the programmer does not explicitly poison the output channels of a process before termination, Flow will automatically clean up and poison such channels.

3.6 Built-In Functions

The built-in functions in Flow are described below:

- `print_string(string x)`: prints a string x with no newline appended
- `print_int(int x)`: prints an integer x with no newline appended
- `print_char(char x)`: prints a character x with no newline appended
- `print_double(double x)`: prints a double x with no newline appended
- `println()`: prints a newline character
- `rand()`: generates a random double between 0 and 1 inclusive

3.7 Program Structure

At the top level, a Flow program consists of global variable declarations, function declarations, and process declarations.

```
program :=
    decls EOF

decls :=
    decls declaration_stmt
  | decls function_declaration
  | decls process_declaration
```

The entry point into a flow program is the `main` function. The body of this function may call a series of procedures, perform computations, and, most importantly, define channels and invoke processes with those channels. Invoking processes with channels establishes concrete links between processes, creating the KPN.

3.7.1 Control Flow

3.7.1.1 Selection

We use if-else statements to selectively execute blocks of code. An expression within an `if` clause must evaluate to a boolean. An `if` statement does not need to be accompanied by an `else` statement.

```
if (conditional_expression) {
     print_string("This will print if the condition is true.");
}
// optional else clause
else {
     print_string("This will print if the condition is not true");
}
```

3.7.1.2 While Loops

The `while` loop contains a condition that is a boolean expression and executes the list of statements in the body of the loop if the expression evaluates to true. With each iteration, the condition is re-evaluated. If the condition evaluates to false, the while statement terminates and the loop body is not executed.

3.7.1.3 For Loops

The `for` loop performs iterations of a block of code and is of the following form:

`for ( expr_opt; expr_opt; expr_opt ) { stmt_block }`

`expr_opt` is an optional expression. The first `expr_opt` is executed prior to entering and executing the statement block. The second `expr_opt` is the condition that needs to be met for the statement block to execute. The third and final `expr_opt` is executed at the end of every iteration.

Variables used in the optional expressions must be declared before the for loop.

```
for (;;) {
     print_string("This loop will not terminate.");
}

// i must be declared outside of the for loop
// loop prints: 123
int i;
for (i = 1; i < 4; i = i + 1) {
     print_int(i);
}
```

3.7.1.4 Continue Statements

The keyword `continue` can be added in a `while` or `for` loop to prematurely finish an iteration of the loop. The loop will continue on with the next iteration if the appropriate conditions are met.

3.7.1.5 Break Statement

The keyword `break` can be added in a `while` or `for` loop to prematurely terminate and exit the loop.

3.7.1.6 Return Statements

The keyword `return` can be used in functions and processes to return control of the program to the calling function or process. If the function has a return type, an expression of that type must come after the return keyword. Return statements in processes may not return a value. They are simply used to terminate the process.

3.7.2 Scope

Scope in Flow follows the same semantics as C. There exist global scope and block scope. Globally scoped variables can be accessed anywhere in a program but never reassigned to. Block scoped variables exists in blocks (compound statements) such that variables declared within a block are accessible within the block and any inner blocks. If a variable from an inner block declares the same name as a variable in an outer block, the visibility of the outer variable within that block ends at the point of declaration of the inner variable.

3.7.3 Creating a KPN

3.7.3.1 Processes as Nodes

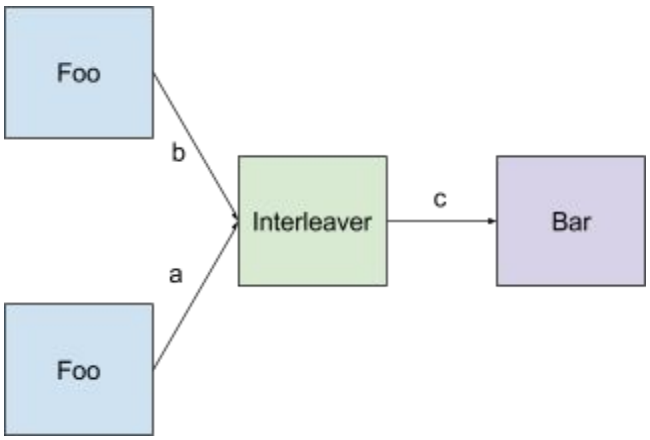
A process definition is a blueprint for a node in the Kahn Process Network. A process invocation creates and starts a single node in the KPN. A process may be invoked an arbitrary number of times, producing a corresponding number of nodes. Process invocations can occur in any block of code.

3.7.3.2 Connecting Processes

Connecting a process amounts to passing it the appropriate channel arguments in its invocation. The action of connecting two processes with a channel results in two connected nodes in the KPN.

Suppose process `foo` takes a single `out int` channel argument and process `bar` takes a single `in int` channel argument. Then, these two processes can be connected with `interleaver` to create the KPN pictured on the right:

```
channel<int> a;
channel<int> b;
channel<int> c;
foo(a);
foo(b);
interleaver(a, b, c);
bar(c);
```



3.7.3.3 Channel Binding

As per the definition of Kahn Process Networks, channels must be connected to exactly one receiving process and one sending process. This restriction is enforced at runtime. At runtime, the first process that writes to a channel “binds” to that channel for writing. If a different process ever tries to write to that channel, a runtime error occurs, and the program exits. Equivalent binding rules are enforced for reading.

4 Project Plan

4.1 The Plan

At the beginning of the semester, our team set up regular meeting times twice a week -- Sundays were our primary work days and Wednesdays were our meetings with Professor Edwards where we discussed the design challenges of our language. The idea for our programming language was conceived in the first few weeks of the semester. Our original goal was to create a language that would be helpful in solving problems we had encountered -- specifically problems of processing streams of data and logging. This led to our discovery of Kahn Process Networks from which the Flow programming language was conceived.

We worked steadily throughout the semester. The “Hello World” deadline and the regular meetings with Professor Edwards helped keep us on track. We planned to have deliverables every week and were able to make consistent progress. We first created a test framework and built a pipeline from the scanner, to the parser, to the semantic analyzer, and finally to the compiler. Then, we were able to add features and expand the capabilities of our language to support full Kahn Process Networks.

4.2 Testing

We set up a testing framework based off the MicroC example test suite. The test suite had to be updated so that it not only supported tests that should succeed, but also supported tests that should intentionally fail. We initially started to write tests for the parser but realized that it would be more productive to write end to end tests. Upon making this realization, we strove to get the foundation of our compiler working as quickly as possible. Working backwards by writing broken tests that should be working became a very effective workflow. Originally we used github issues to monitor what needed to be done, but many of our issues were overly broad, such as “clean code,” which had no foreseeable future of being completed. Group members would write tests trying to break Flow, where other group members would fix the broken tests. Unlike github issues, which could be ignored, these broken tests had to be fixed.

4.3 Style Guide

We used the following guidelines when developing our code:

- Each line of code should remain under 110 characters
- Use block comments to annotate code
- Write utility functions for frequently reused code
- Use underscored names rather than camel case
- Use consistent indentation

We made sure our code style was consistent by running our Ocaml code through Camlp4 and our C runtime environment code through clang-format.

4.4 Software Development Environment

The following are the specifications of our development environment:

- OCaml 4.02.3 - for development of the scanner, parser, semantic analyzer, compiler
- clang-700.1.81 - for compiling our C runtime code
- graphviz 2.38.0 - we specifically used the dot tool to generate our Flow graphs
- Github - for version control

We developed and tested our code on Mac OS X.

4.5 Timeline

The timeline below gives a general outline for the development of Flow.

Date	Item Complete
9/30	Project proposal
10/26	Parser written
11/1	AST with conflict-free grammar
11/3	Pretty printer
11/3	Test framework set up
11/15	Interleaver (Hello World) working. Compiler without semantic analysis built
11/20	Semantic analysis integrated with compiler
12/16	Bitonic sort demo program complete

4.6 Roles and Responsibilities

We each took on a mix of roles and responsibilities for this project:

- Zach - Tester
- Hyonjee - Compiler backend, C runtime
- Adam - Language guru, bitonic sort master
- Mitchell - Compiler backend, C runtime, semantic analysis, dot translation

4.7 Project Log

The project log below shows 220+ commits beginning on September 13th.

```
commit 7a45fc134dfe6b7989590f2b6fff2e8c4bada2a2
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
```

```
Date:   Thu Dec 17 20:45:27 2015 -0500
    fixed formatting

commit 2b47167372a1f64bbbec6bb4ab19883b4bfdc8be
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Dec 17 16:30:09 2015 -0500
    Added license

commit 97c3a41b793b7dec89605ed106f9687bbb04c19c
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Dec 17 16:09:19 2015 -0500
    added dot printing, cleaned up auto-poisoning

commit 8dd7e7effdb0fa3be5c71b98b18e8c6efef23d58
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 22:15:01 2015 -0500
    refactored boilerplate.ml to c_runtime.c. boilerplate.ml is now generated by make.

commit eee16b7c4a0e6e332598f32338518f8a7769e3e4
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 18:27:20 2015 -0500
    channels are automatically poisoned when their writing proc returns

commit 14e2031406e0f341343954df9828b18febc8acb5
Merge: 198a115 33a09fb
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 17:27:57 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit 198a11568272e5b79077613128a05a9819d31461
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 17:27:53 2015 -0500
    added names for runtime enforcement of single channel reading/writing

commit 33a09fbb9a382f7263664dca22bc8e5cb82c8e72
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 17:09:29 2015 -0500
    fix some broken tests

commit 3fac3d7d3ce602394b2f5b5cb39716f385847d26
Merge: 33f1e70 0a818c4
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 16 17:05:03 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit 33f1e703240763ff4a7d33323a6e299d005c6b7b
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 16 17:04:37 2015 -0500
    Working bitonic sort for arbitrary input size 2^n for some n!!!

commit 0a818c47c708d44b40082c2c1f24538c83fe2ec9
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 17:00:13 2015 -0500
    fixed broken tests, added enforcement of single-channel binding

commit 2bb08a940f25f35386296923ebe8eeb348786300
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 16:16:59 2015 -0500
    fixed broken chan test

commit 379b279738eafc29da88e45ae44c8ba14237eb0b
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 15:51:39 2015 -0500
    fix bug so that that test fails

commit b41bdc13fa990951528d9303b01f4bd1a9b0b336
Author: Zachary Gleicher <zachgleicher@gmail.com>
```

```
Date:   Wed Dec 16 15:48:32 2015 -0500
    add tests for single input and ouput for channnel

commit 174bcbad02a2ba98da19025f5b8a8043cf1cbfe0
Merge: fe02e0c f769871
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 13:48:11 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit fe02e0c9186aefa7656b49ea07edc4f4ddd944c5
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 13:47:57 2015 -0500
    Added capability to return within process

commit f76987116c34f2e321c4d2b379f3054c601f27ee
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 13:27:37 2015 -0500
    add test to check that channels of channels and lists are not supported. test that in chan cannot be
poisoned

commit 33e0606c584d86f4e4c7158d233ccbdb3a1b7301
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 12:58:44 2015 -0500
    add test for return in proc

commit 740626c8f364d702c6125579ec2c260c19926f20
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:57:50 2015 -0500
    cleaned up parser

commit f16a049fbcbf440dc52e2575fd7d4dd21a3cb9de
Merge: 5496a51 089404a
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:54:36 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit 5496a51e434b7d83c5248c82489e5a236a7e8688
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:54:30 2015 -0500
    directionless channels as args are compiled properly

commit 089404abf2d4a96d62378e9d3afcfb949a6e89ad
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 12:51:01 2015 -0500
    add test for passing directionless channel into function

commit d05b092ff49e4e7171e14c9f16f80fb7749079e6
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:50:18 2015 -0500
    directionless channels can be arguments

commit 8dadba2c1590f38252f1de2b15f1207a3ce02a04
Merge: a525ce9 369f3bc
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:46:34 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit a525ce9398c3e7394f894ee2622c807e7468bffa
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 12:46:29 2015 -0500
    fixed return semantics

commit 369f3bc1fa62299690de0657e844db69620b661b
Merge: 270b532 9bc3639
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Dec 16 12:25:19 2015 -0500
    merging scope tests
```

```
commit 270b532fa09821a36f5b349308362250b8c6ca7c
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Dec 16 12:24:57 2015 -0500
    removed len built-in function, we're using # operator for list length

commit 9bc363919748997e0448741413d2e6af810ca085
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 12:22:27 2015 -0500
    add tests for scope

commit 88db94b977bdf3cb52994d9da430b8926603c329
Merge: e4feef5 dd56125
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 10:45:54 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit e4feef531d24e40f3ec175468f765a5b33ca6aff
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 10:45:09 2015 -0500
    fixed failing channel return test

commit dd561251442ef91667320cd1c3c240a6909d71b4
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Dec 16 10:28:17 2015 -0500
    removed structs from flow

commit 72e18ccb769a300a36a715db9d0fa0c4323b7b54
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 10:18:58 2015 -0500
    remove bad tests

commit b31c7549e359c39c38287783835e89d11eaa09d5
Merge: 2669933 cd4c95a
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 16 02:19:42 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit 2669933562e1b60727ff91c2092790754d28feb5
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 16 02:19:23 2015 -0500
    Add working bitonic sort (no automatic mux/demux, occasional hardware fault)

commit cd4c95a766407c79c3b7edbf42d54270d3a3e724
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 16 01:04:59 2015 -0500
    strange multithreading happening with test

commit 1c288504b5c49f6bf84be42d23939ee539d1ca24
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 16 00:16:53 2015 -0500
    Added more tests

commit 7f1d56adb3675fc5ee9f656b8ca79b7499331eed
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Dec 15 22:40:59 2015 -0500
    fixed bug that prevented no arg functions

commit 19592caa8c557865f3889c027942a5b8a245a952
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Dec 15 22:24:04 2015 -0500
    we can now return lists. Also fixed reinitialization of lists

commit 55601b6892335118b7605a987acf026bb187e164
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Dec 15 21:06:58 2015 -0500
    fixed scoping problems and global variables
```

```
commit ffb0ce29878f569d2c7be176a9c704e4e557187a
Author: hjoo <hj2339@columbia.edu>
Date:   Tue Dec 15 19:22:13 2015 -0500
    removed bit shifting tokens and change concat to right associative

commit 1cfdbfc3a4a57ff738e10bb778362e34f9ef7adc
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 15 14:56:37 2015 -0500
    add failing tests for functions that returns lists and channels

commit fab1fc2d59d3501208d4790aa0cd40a7fcf34136
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 15 12:36:27 2015 -0500
    add tests for scope

commit e7c9059f07ea241d6b0ab702126a17a92b184a47
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 15 12:13:44 2015 -0500
    failed test for initializing a list

commit 3c89a6b584bf2bc94e0a70470cc3bd7392026189
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 15 12:07:08 2015 -0500
    add test for accessing head and tail of empty list. Add test for tail of one element list

commit d563029dc6c30f58866d2d785252c8629e92c23f
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sun Dec 13 19:53:59 2015 -0500
    took away assertions

commit 07969b6b3014030147b3bf50b1ac498b1a209a3e
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Dec 13 19:33:48 2015 -0500
    add initial bitonic

commit 87c7487fefed7bf86429f15e188a35cf1da6e2f6
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sun Dec 13 19:10:43 2015 -0500
    mission abort on reference counting. It currently causes a deadlock

commit 25a8944682c493e102b4f3fcd729d83816d7bea4
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sun Dec 13 18:53:34 2015 -0500
    fixed concurrency problems with lists. added tests for channel lists.

commit 7e510f0070d8f483cda0e23e5467b6c132ea1424
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Dec 13 16:31:32 2015 -0500
    add test to ensure failure if tyring to read from poisioned channel

commit cf283757712d43b3aebcc57d87b9156c7fe111f2
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Dec 13 16:29:17 2015 -0500
    refactor test suite so that it does not check outupt of programs that should fail. Add check to see if
fails at runtime. delete fail.out files

commit 542c2574f06365cce5706fc9a98678caff043b64
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Dec 13 16:19:55 2015 -0500
    added running sum test. using this example as hello world for langauge tutorial.

commit 9877499bc95e1b104b6c116f7e8cb5daf8f7f734
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Dec 13 15:33:11 2015 -0500
    flush stdout after each call to print function
```

```
commit 58fdd25015e5f36edcecb146bb76dcfaebc3130d
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Dec 12 14:39:40 2015 -0500
    got lists of int channels working

commit 8ed8f6e041800a3955fe7da55e807dd222b65083
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Dec 12 02:51:44 2015 -0500
    cleaned up thread list. It is now a simple list that grows from the tail

commit a5e15498a65058306e336decbe03ff97c732be77
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Dec 12 01:52:06 2015 -0500
    removed structs from parsing. All tests pass

commit 03a19fa5eb098a1364f6f2d91a3451f9332b0fa5
Merge: b90e179 9e3ecb0
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Dec 12 01:45:05 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit b90e179940ec945218a10a65180247e379124ccd
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Dec 12 01:44:43 2015 -0500
    debugged absurd race condition

commit 9e3ecb0f045fe16750eca3751114b1357c33811e
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Fri Dec 11 18:53:29 2015 -0500
    add tests for bad declration, char lists, double lists, and string declaration

commit d55319ec6c12f2f33deb380bf1efe1872842a5dd
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Fri Dec 11 18:10:58 2015 -0500
    add test to init with empty list

commit 5cdf7d4cc1feaebb4c567e96042db6f9adb25a7c
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Fri Dec 11 18:05:37 2015 -0500
    fixed order of function args - all tests pass

commit 25839cda55b28c9caf5f9ae828b513d14ab75f84
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Fri Dec 11 16:27:11 2015 -0500
    proc-in-proc works

commit 7405cf440f246b1befefa96270613c1e3cc33b06
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Fri Dec 11 00:46:00 2015 -0500
    fixed list assignment to make it stop segfaulting

commit a744023aea99f3c70cd1246431a05b2a021dc2c7
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Thu Dec 10 00:05:50 2015 -0500
    add a bunch of tests for lists. Fix bug with list initialization

commit 28ae457c27d3d72d666b0c9c31bdb8cc82b19796
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 9 13:32:25 2015 -0500
    add test to check that channel can be passed into function. Currently failing

commit 263b5981239ca3bbe52a1636be5eac2629d6d29f
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 9 13:27:23 2015 -0500
    add test to call process in a process. Currently failing

commit a27a5905aa6ce7399843ef3b17b13906d7aee938
```

```
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Dec 3 02:05:39 2015 -0500
    added reference counting for reassignment of lists

commit 4049db54e2aea0910c802f3e8d5d0a38adcd35d4
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 9 11:05:09 2015 -0500
    bug fix in boilerplate so that make enqueue returns

commit 3485e4a18dfbd7d073298c5e54e865380d364cb7
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 9 10:51:17 2015 -0500
    write test to check that list can be passed to function

commit 9885e15572a870129de952e3f2cfdd015cb46276
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 9 10:47:23 2015 -0500
    add test to check that list declared in correct order

commit 0626ed19cab3977213c1536eda1acc6ec40bf6b6
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:57:21 2015 -0500
    modify test script so deletes intermediate error message files

commit 876da5ce361ed14d0f18bd5897f6e09ae8251dfa
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:54:33 2015 -0500
    add annotations for failures

commit 433363e9fdfabace87dec801bb93aa41bf38df53
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:53:36 2015 -0500
    add test to ensure that you cannot writ wrong type to channel

commit 1f08fedcc6c35887072b2f0d54d42f795edf5795
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:44:32 2015 -0500
    write test to check that you cannot write to an in chan

commit cf15adcecfd6cac6fa680d3cd40073ba0bc72b50
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:40:12 2015 -0500
    add test to ensure that you cannot read from out chan

commit d19dc841639236b51c5eaae058227047714395ec
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:34:11 2015 -0500
    add a test for recurion

commit 8bf06409fea1df03baf7bd18ce8622cc81a71652
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 18:18:47 2015 -0500
    add test for undeclared function

commit 4b19d7053add80850fc9f64327deb262595a4302
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Dec 7 12:10:24 2015 -0500
    fix sum test

commit 503422c17d0b5b6b8135eedf9f5781609571c6ab
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Dec 6 19:01:33 2015 -0500
    add sum test

commit 313eeef2b88f5f75e6ece817d98dde3033ea9b47
Merge: 44de4dc 226b189
Author: hjoo <hj2339@columbia.edu>
```

```
Date:   Sun Dec 6 18:50:40 2015 -0500
    committing new pulled files

commit 44de4dc48fd83734e5249e1fdfdd21fee9a407ac
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Dec 6 18:50:02 2015 -0500
    basic animal farm demo program worksgit status! new issues filed for bugs discovered - i.e. global
variable decleration

commit 226b189815fba5470787dd20d64e7f6c36414d53
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Dec 6 17:56:40 2015 -0500
    add random number gen

commit 323af1529e36c8cfb93985e9cf7e0906bb4a4184
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 2 15:19:37 2015 -0500
    Fix write_channel precedence

commit 76f1a80a7672ca9882742646a22073d215b418c5
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 2 15:08:14 2015 -0500
    Add fibonacci test

commit 31ce8ee2c30d2308ef56e14090805cb3b9cc6b0a
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Dec 2 14:57:51 2015 -0500
    list initialization passes

commit cd8d7351950697eca3c91f8b9d77865ad0849f80
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 2 14:11:59 2015 -0500
    Add fibonacci test and lower precedence of channel write operator

commit 63ee7fcc5f0004d674519f70dd4ae6ba2e1788b9
Merge: d374c19 a649d86
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 2 12:12:23 2015 -0500
    Merge branch 'master' of https://github.com/mgouzenko/flow-lang

commit d374c194b5b600e6349cb2ef243c4804c319929f
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Dec 2 12:11:37 2015 -0500
    Add a string comparison TODOw

commit a649d861eccec8e2463eafc726e80902d27452fc
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Dec 2 00:03:39 2015 -0500
    add ability to test for semantic analyis error messages

commit 99a2fa9bc545fe9adfcef240c40e153af95dcbae
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Tue Dec 1 23:30:01 2015 -0500
    Add a list length operator (#) and change comments to begin with //

commit a7099998deb4004f63e6dfeba5b4ba9f06975e25
Merge: 9819df9 48cfe3b
Author: zgleicher <zjg2102@columbia.edu>
Date:   Tue Dec 1 22:37:38 2015 -0500
    Merge pull request #27 from mgouzenko/print

    Revert back to separate print types.

commit 48cfe3bc4a0c9b942d48ee4f39bf2a8ae0dd2a21
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 1 22:35:23 2015 -0500
    remove len function
```

```
commit 730f50ae70c667a83a16e88f11b9af75cefa0a99
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 1 22:34:51 2015 -0500
    remove special check for built in types

commit c4f544c0cb9b8df355905b8541725468d668f992
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 1 22:28:29 2015 -0500
    Fix print so that it does not use variadics

commit 05157160a842dbee58e2ba0c428f310044cd9700
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 1 22:10:41 2015 -0500
    Revert "add print and println function so that any type can be called"
    This reverts commit 392d4b3a6a02279eefd58ad4866ce68f8eddef8d.

commit 9bb1a1fe89f6ddf683f5e7265f544c5800963082
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Dec 1 22:10:06 2015 -0500
    Revert "remove old print functions"
    This reverts commit 9819df9ccfbbc95a2b00c3787365c0df71553916.

commit 9819df9ccfbbc95a2b00c3787365c0df71553916
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 30 19:13:00 2015 -0500
    remove old print functions

commit 392d4b3a6a02279eefd58ad4866ce68f8eddef8d
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 30 19:00:46 2015 -0500
    add print and println function so that any type can be called

commit 9131e0410877f9239841fb955326c340e9ce81f5
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Nov 29 17:09:01 2015 -0800
    add test for function call

commit 8defc78dc044f2c837f155a5dc49ac6fcf8841fa
Author: hjoo <hj2339@columbia.edu>
Date:   Mon Nov 30 17:38:22 2015 -0500
    length works with lists of different types

commit 6cf8b65d815ba61173ba1e7027fb418a095aea43
Author: hjoo <hj2339@columbia.edu>
Date:   Mon Nov 30 16:27:13 2015 -0500
    added built in len() function for integer lists.

commit 241fcbe9de8493704e8ec375f4be295ba29cfdf4
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 29 18:42:37 2015 -0500
    list add front test is passing. had to fix the _get method to account for wrap around.

commit 23c66b486328ffcc42f21efa060c4b0414fcbfad
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 29 18:31:30 2015 -0500
    add front not working for lists longer than 4. added tests to test add front.

commit ddc0a2935262b596696dd5600e785f5a80308ead
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 29 18:28:26 2015 -0500
    working on list initialization

commit b7df08894a02c94dbfa851e838f794d36d991317
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 29 17:16:14 2015 -0500
    forgot list tests from last commit.
```

```
commit 7467da59971e0865b09987f4f7e34a767188ccaa
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 29 17:14:34 2015 -0500
    fixed boilerplate code for lists so that lists of length longer than 4 work. added test-list2 which
tests list initialization. list initialization still needs to be implemented.

commit d8b42e3aa9891ac5414fffc84f186a52024086fd
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Nov 28 01:08:06 2015 -0500
    removed lgc flag so that the tests pass

commit e25e0fc44c48f83febc4113b71f169fed03d26ce
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sat Nov 28 00:58:28 2015 -0500
    All tests pass. Basic lists work.

commit 02a238a1e7dfc8e17f32e1655c5a7cdca809a870
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Fri Nov 27 23:03:39 2015 -0500
    Double channels work. Arrays are gone. Lists parse and translate

commit c627e4100c76243262821d89d5c647d5ad0a36be
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Fri Nov 27 02:29:02 2015 -0500
    refactored all duplicate code into macros

commit 45ec6f9dee840abc75181304fc6df71c4ef21723
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Nov 26 22:37:42 2015 -0500
    fully generalized dequeue and enqueue to macros

commit d4d52f5b6c0a61e309ecef14d2a2a521df8b2eb4
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Nov 26 17:32:53 2015 -0500
    cleaned up compile, refactored boilerplate code with macros

commit 76dbdd043503b177aa127bd997f2a27b2c71cb0e
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Nov 26 14:52:07 2015 -0500
    fixed a few bugs, commented the sh-t out of semantic analysis

commit 93dfb8369d4419c273b0ff60109274c4480eff59
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Nov 26 02:11:08 2015 -0500
    Added some comments in semantic analysis, removed list.rev

commit 8383137a53c4cead05aa7f3583fda3e1e83bdb20
Merge: fff1869 f39f85b
Author: hjoo <hj2339@columbia.edu>
Date:   Tue Nov 24 18:48:48 2015 -0500
    merge char tests

commit fff1869cd6b9ff24cadec745223c1701e511f6b9
Author: hjoo <hj2339@columbia.edu>
Date:   Tue Nov 24 18:48:19 2015 -0500
    separate out boilerplate c into it's own file

commit f39f85b44d264d663860322d1a3a08b43e84488c
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 18:30:17 2015 -0500
    make char interleaver slightly more compliated

commit 88b6fc8945b4b1d6aabb80aab40718d48e10896b
Merge: 14e6261 cfb3b33
Author: zgleicher <zjg2102@columbia.edu>
Date:   Tue Nov 24 18:13:46 2015 -0500
```

```
    Merge pull request #16 from mgouzenko/char

commit cfb3b3306022492cda0d6c0794e6daa69a9f38b2
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 18:13:12 2015 -0500
    add test for char interleaver

commit 025780542766b402071b0dcabd78f9fbf2a15e26
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 18:10:46 2015 -0500
    char interleaver works

commit 516f7276d315b191b30ee5c69f1061197125fd85
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 17:56:39 2015 -0500
    add channel type so that it is not hard coded in translate_vdecl

commit dbee83f7b563e2b511690ed5b3c4e7ab440c507f
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 17:42:47 2015 -0500
    ensure that proper type is enqueued on channel

commit a2d0ac93a16f1abad1bbbf5465b38b2a96714cea
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 17:35:21 2015 -0500
    make deque channel use correct type

commit 43680de779fadf4616f9d8a10082c20c28a8c316
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 17:21:29 2015 -0500
    small bug fix where so that type of expression in a being poisoned should be a channel

commit 3a3e01e084b197b263e85bc9b9b5c6443a4bb083
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 17:16:23 2015 -0500
    update wait_for_more and poison so it can handle correct channel type

commit 14e6261071e6d102c277487fba12c6fa35725894
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 24 16:35:11 2015 -0500
    add c source code for simple char interleaver

commit dd26fedecd2f97fe42f2f675fff40afcb7da54fc
Merge: 9f45655 562d5b1
Author: zgleicher <zjg2102@columbia.edu>
Date:   Mon Nov 23 20:12:01 2015 -0500
    Merge pull request #15 from mgouzenko/semantic_analysis

commit 562d5b1fe1a19f8fb84b820eb1e0c7045c798850
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 23 20:11:14 2015 -0500
    program decl and args struct for process call needed to be reversed

commit 9fe2fd2ea5ffa5700d19d76d701c6c125baea164
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 23 18:46:32 2015 -0500
    remove ^ token so that semantic analysis can resolve if a channel called in a conditional and add
_wait_for_more if necessary

commit 1c5bcd961d285a86c06fbf8c13deeeb2269a1f83
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 23 16:06:38 2015 -0500
    check if channel being called in conditional

commit 9f456554f8d9af542ec8df9abe226bd2dc0b61e0
Merge: a66981d 8e501fe
Author: hjoo <hjoo@users.noreply.github.com>
```

```
Date:   Sun Nov 22 17:46:03 2015 -0500
    Merge pull request #13 from mgouzenko/semantic_analysis

commit 8e501fe0ec7bd0e1685173a8db6113f10958ef43
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 22 17:44:28 2015 -0500
    fixed process call parameter checking, removed ~ symbol, fixed function calls with multiple
parameters, changed tests to use new flow print statements

commit d322d4f632a9d0bee3d2ee15574bd74c5130caf9
Merge: 737ea05 a66981d
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 22 16:08:18 2015 -0500
    print tests added

commit 737ea05c92d5633a5838b4fbf288d2861fa82744
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 22 16:06:27 2015 -0500
    removed Float type, added different built in print functions for newline

commit a66981d49f2a2021f6db1e9906e4bc1c4b0c7d57
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Sun Nov 22 16:02:54 2015 -0500
    add tests for printing

commit e6f0bca1fd5a94aadb077c27c40668487f8b04bf
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 22 15:35:09 2015 -0500
    printf in flow translates to printf in c, removed Float type in flow_types

commit 1233a684f6260e511340a6517aa384cebe04dc4e
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Fri Nov 20 15:38:09 2015 -0500
    integrate semantic analysis with compiler

commit 5e2ad85fff950a6b39122a64e98539274953640d
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Fri Nov 20 14:36:04 2015 -0500
    adding sast pretty printer which i forgot to add in the last commit. Probably not going to focus to
much on the pretty printer

commit 3e1c4eef59bc762dfb3d6eb4dbf4c67a789ccba5
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Nov 18 13:16:30 2015 -0500
    pretty printer for sast almost works. Can check for semantically correct programs, but not sure how to
print the type of a wildcard for the tree

commit a0d2ebde6c9948feacfa90c21cd376b9241f8423
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Nov 18 11:56:04 2015 -0500
    set up printer for sast. No code written yet for sast printer.

commit 8a04995a559dadb0f4a0e6ed3e82abd41336ec17
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Nov 18 02:03:17 2015 -0500
    add sast for function call

commit 83450211f849dc5b07d0061019ae36412dc84e7a
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 17 23:06:57 2015 -0500
    add unary op checking

commit 099a3e3bc5250b4b70b473af6fb4ad2da25883b3
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 17 20:12:30 2015 -0500
    Basic groundwork for semantic analysis done
```

```
commit 516d3d6cc13cd3240cd1d78de5c2053d5e3604ba
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 17 02:21:19 2015 -0500
    Added semantic checking for statements. None of this shit is tested

commit 1c0771d59ab4d309c615872611deb02b2fba9dad
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 17 01:32:36 2015 -0500
    added function declaration checking

commit 23089e345bceda51fd034059cfeca2818ef6944f
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 17 00:29:01 2015 -0500
    finished variable declaration

commit dd1c9bbfcb9383dc74fabdfae69a7d7b6aada2f1
Merge: 2f57a44 4c35b6d
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 15 18:58:34 2015 -0500
    merging with master

commit 2f57a44a33702b2ddeef4233b257722c2c570cf7
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 15 18:57:43 2015 -0500
    tweaked some array code and working on binop check of semantic analyzer

commit 4c35b6dada9edb3dc69b865d2773f798c74ca932
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 18:03:06 2015 -0500
    Add simple interleaver to test suite

commit 6c97b50f8abf5036ff6820bb7b9428ef8386d432
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 18:00:23 2015 -0500
    Getting simple interleaver hello world to work (AWFUL code though)

commit 3b0c549a1a8cc21fa0ed90f8c525cc17c4526c3f
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 15 15:59:01 2015 -0500
    cleaned up some small errors in compiling

commit a8099848f167e1348955ef451d1cb4f8dbe41a8a
Merge: 8d8f5be 886dfb2
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:32:48 2015 -0500
    Merge pull request #4 from mgouzenko/boilerplate

commit 886dfb27d02ca9fbda5425f7da0c970c8b856492
Merge: 0e056f1 c56ee48
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:32:01 2015 -0500
    Merge branch 'boilerplate' of https://github.com/mgouzenko/flow-lang into boilerplate

commit 0e056f1d0cb0b4e08055b1e34f7b6ba3f6bf16c3
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:21:47 2015 -0500
    Update the boilerplate globals to lead with underscores, and add Travis slack integration

commit 7fdd147e44f202897ca6116dea4a36f7ffd113ac
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 21:48:08 2015 -0500
    add boilerplate code

commit c56ee48e7898a308eeb3b8ed8f4f45c1203bb7e1
Merge: 0f59ee4 4c14cb0
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:23:47 2015 -0500
```

```
    Merge branch 'boilerplate' of https://github.com/mgouzenko/flow-lang into boilerplate

commit 0f59ee48c20352d5a5f69549286caf17eed2be16
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:21:47 2015 -0500
    Update the boilerplate globals to lead with underscores, and add Travis slack integration

commit c15278764fe7c4c1cf6dae8cf6bf45f42d02ecbe
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 21:48:08 2015 -0500
    add boilerplate code

commit 8d8f5be10ff28ceb96ab8396e28925f5e1d8007d
Merge: 728fd5f 303739d
Author: hjoo <hjoo@users.noreply.github.com>
Date:   Sun Nov 15 15:22:17 2015 -0500
    Merge pull request #3 from mgouzenko/expr

commit 4c14cb0c754576dbaafadf7204f9c9419b061f53
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 15 15:21:47 2015 -0500
    Update the boilerplate globals to lead with underscores, and add Travis slack integration

commit 303739d06150f0929c397d173502a43e3dcb9c0e
Author: hjoo <hj2339@columbia.edu>
Date:   Sat Nov 14 14:05:21 2015 -0500
    started semantic analysis and fixed minor malloc bug in compile

commit 749d9a70d65b0c5afb01486d7f87795bd865e742
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Thu Nov 12 11:27:20 2015 -0500
    fixed array

commit 5a1e86819d3fb0e1c9c719b1f529b1542a2d9b01
Author: hjoo <hj2339@columbia.edu>
Date:   Thu Nov 12 10:02:18 2015 -0500
    matched array var names in test-array1.flow

commit c34a87447fe1c999f9e84049917a0baabaa30932
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Thu Nov 12 00:51:58 2015 -0500
    trying to compile array. Error with identifier

commit 9f1518b54f80c5be5acf7814b4b7f8c30d5388b3
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Thu Nov 12 00:04:19 2015 -0500
    compile arrays

commit 728fd5f6e8f1badadaf15032979c9ca10ee20d29
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 11 12:52:35 2015 -0500
    Add poison, array element access, and array initializers to the grammar and AST printer

commit 89c3da762ffa3d6425abbe3609c196c733087235
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 22:34:58 2015 -0500
    add tests and fix boolean bug

commit 357f782255cb4211d60865994a39e31676abb8e2
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 21:48:08 2015 -0500
    add boilerplate code

commit 87d56516724efaf34e4b78f6be95d2e4de123a42
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 19:56:28 2015 -0500
    set up regression tests for c complilation
```

```
commit 30e4cce4f909a24ba05a80e0a3562c6fa5315cf5
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 19:14:46 2015 -0500
    fix stmt list so comma is not at end

commit 773ab0a684535199a5023f1deca53cd8067e825e
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 18:00:44 2015 -0500
    update regression testing framework

commit 87d1059d4dd82fb585407a4083c36e6844584edc
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 17:22:39 2015 -0500
    function expression compilation

commit 2f270dceea21853d62c643b1f8ec6fd1096bd65d
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 01:58:32 2015 -0500
    simplify unary operator for c compilation

commit 3bb56121a3c59ddca87f127363c2dd4367244fab
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 01:52:39 2015 -0500
    update send to channel for c compilation

commit 9e3e43e8b68ac4439c139469222c070490a2759c
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 01:01:53 2015 -0500
    fix retreive so that function is called

commit 9e0908b7bb1560f69802fa0a358b8f80ad3b752c
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Nov 9 23:57:39 2015 -0500
    Started work on compiling expressions. Still need to do FunctionCall and StructInitializer

commit 72a5d0ae81f05076c00bcd7f4918a90eae1a1190
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 10 00:11:17 2015 -0500
    added statements

commit 198b945bb632384a8c35fb103f908ce205e32f5f
Merge: 9a168f1 72875cf
Author: zgleicher <zjg2102@columbia.edu>
Date:   Tue Nov 10 00:05:37 2015 -0500
    Merge pull request #2 from mgouzenko/star

commit 72875cf2ca4393fc3a6f23145387903598ff93f8
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 10 00:04:04 2015 -0500
    add ^ token which is a placeholder for wait_for_more. It will be removed after we have done semantic
analysis

commit 9a168f122e6eec1275a08e1661d9321e771b51d6
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Mon Nov 9 18:59:48 2015 -0500
    Added translation of variable declaration.
    Also added unpacking of process arguments inside the body of
    the process.

commit 3f1118f12d025237c3b3a4bbc112af925e194124
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Mon Nov 9 14:44:34 2015 -0500
    Added the beginnings of the actual compiler

commit 25fdb12b5ff9ef4f56694cc421cdbbe9f203ad9b
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
```

```
Date:   Mon Nov 9 00:27:21 2015 -0500
    Added global threadlist. Refactored flowc.ml.
    Threadlist keeps track of threads in c version of
    interleaver. printer was factored out of flowc, and
    compile.ml was introduced.

commit 106297d0ae96b51495735e875f0cc8e028247522
Author: hjoo <hj2339@columbia.edu>
Date:   Sun Nov 8 18:55:24 2015 -0500
    int interleaver in c worksgit add int_channel.c int_interleaver.c ! some things are hardcoded though.
i.e. joining threads and size for tokenizing arrays

commit 73371367007710ab4871648cf7978698788074a1
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 16:56:43 2015 -0500
    Add lists and arrays to grammar, AST, and pretty printer, and add a test

commit 6040c5d6786b30592147fef87e965d0dc9d6af66
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Wed Nov 4 12:55:05 2015 -0500
    added int channel

commit 9362561df1429e7d4e802948f8dd141a93735b40
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 12:52:17 2015 -0500
    Update Travis status

commit f5815d44a3a190ce98535bcb22575246fedd8c6d
Merge: a38da7b 6f30770
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 12:49:48 2015 -0500
    Merge pull request #1 from mgouzenko/flow-parser

commit a38da7beadcd8e90a21256f096a29481edc5ede8
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 12:46:57 2015 -0500
    Add Travis status to master README

commit 6f3077087e51a25da42dca43d469f06bbc958523
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Nov 4 12:45:05 2015 -0500
    no printf in flow hello world

commit c9c89b28aee80e4ce1a80162f798b609a528d6d7
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Nov 4 12:43:17 2015 -0500
    hello world v1 in flow and c

commit 6ea5a25951728e799609c4c3d50f8a190ef14f13
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Wed Nov 4 11:30:50 2015 -0500
    Identifier cannot start with a underscore

commit 3d1ec6d32f1fee56020f2dabb1e563527cf42853
Merge: 5042d9f 5e3a2da
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 10:53:13 2015 -0500
    Merge branch 'flow-parser' of https://github.com/mgouzenko/flow-lang into flow-parser

commit 5042d9f98901d441815189469c6aa5eb1350c70d
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Nov 4 10:53:02 2015 -0500
    Allow function declaration without definition

commit 5e3a2da022763363a8b913e39ffd2232627eeaee
Merge: df2d423 4e01c9d
Author: hjoo <hj2339@columbia.edu>
```

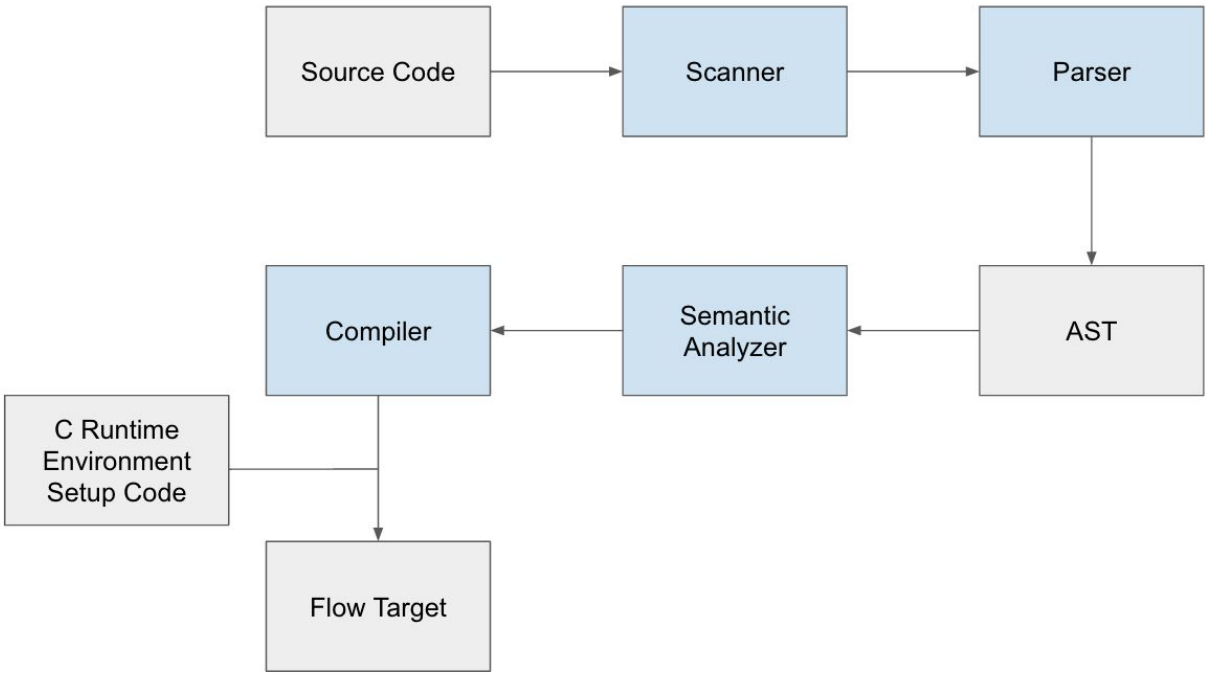
```
Date:   Wed Nov 4 10:29:55 2015 -0500
    removing mc tests

commit df2d4231ab742c04a8045c54c91bd5d8187fdd4b
Author: hjoo <hj2339@columbia.edu>
Date:   Wed Nov 4 10:28:59 2015 -0500
    getting rid of mc tests

commit 4e01c9dc54e268f8ae03cbce7930e505e8710ffd
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Tue Nov 3 23:33:24 2015 -0500
    Write a test program

commit e4f23eab9c666e6a027df56af06e7ba6ac0dfa0c
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 3 21:54:20 2015 -0500
    travis fixed

commit 407d256f77ed5fd52580f465a864311bf23a6429
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 3 21:50:48 2015 -0500
    travis bug fix

commit 675c6d2499fdaed75fbfe41d8e562c6af501ccac
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 3 19:27:29 2015 -0500
    add travis ci

commit 5cd57b6434f9a40240e3896a6750103e4dcb1351
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 3 18:51:14 2015 -0500
    testing framework for grammar set up

commit 764e4be630f82f06c429f499d3e7ef3d36a3a34a
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Tue Nov 3 13:58:03 2015 -0500
    delete bit operators

commit cee07c8de5824f9ee58af2f29c313f5a0d6b3c25
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Tue Nov 3 01:22:33 2015 -0500
    Finished the pretty printer.
    Pretty printing is done in flowc.ml. The flowc executable will
    now produce two files: out.dot and out.png. The former is the
    AST encoded in a form that graphviz's dot will understand. The
    latter is a picture of the ast.
    To install dot, run:
       brew install graphviz

commit 3501a67c6b821c57880ea93a29607ef7946438b0
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Mon Nov 2 17:38:21 2015 -0500
    made the beginnings of pretty printer

commit 5b36001e99e54320bc33e83e58fe745403a6cd6d
Merge: 6f30e96 3105f24
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Mon Nov 2 14:02:06 2015 -0500
    Merge branch 'flow-parser' of https://github.com/mgouzenko/flow-lang into flow-parser

commit 3105f24da88e751e7000747a3cb45d9e654c5d03
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 1 17:13:51 2015 -0500
    Allow empty compound statements

commit 6f30e96d6c16cfcba5af4132e1e20d6ec3d4020b
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
```

```
Date:   Sun Nov 1 17:10:36 2015 -0500
    added to interleaver test

commit 661a6bdeebbe7ccaaaa8e0194cee9b2bd09df68c
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Nov 1 17:09:38 2015 -0500
    Fixed shift/reduce and reduce/reduce conflicts in grammar

commit 86e41dac182ffe2fe05d327071bc65d86444db11
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sun Nov 1 17:01:23 2015 -0500
    AST nearly done

commit 9d2a1762821e1b03ccf50e884e07fe2481a7dc60
Merge: ac81a1e 89c8da5
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Mon Oct 26 23:48:19 2015 -0400
    Merge branch 'flow-parser' of https://github.com/mgouzenko/flow-lang into flow-parser

commit ac81a1e915a8cba2c7bb706ca6e1d8514f62f704
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Mon Oct 26 23:48:07 2015 -0400
    Fix grammar issues

commit 89c8da535c1c191fa92ac8564430054106cd0a92
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Oct 26 23:38:09 2015 -0400
    add char and string parsing

commit 9bc523a73d6ab0241eaf97e616b3302b74c331fa
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Mon Oct 26 22:58:31 2015 -0400
    No-conflict parser

commit 62d7eb72a8812f8d9486488399d0d89fb9c413c8
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Oct 26 20:48:45 2015 -0400
    starting grammar

commit 2111ef7f3532ebe330353f750e2899dac5bd62fb
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Oct 26 20:28:50 2015 -0400
    add precedence

commit 6c78da5b4d650d802903024f17207c5b540a5a5f
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Oct 26 20:06:24 2015 -0400
    add tokens

commit cc1677d3825831d995d7ee76d225a40edc3ff76f
Author: Zachary Gleicher <zachgleicher@gmail.com>
Date:   Mon Oct 26 18:21:11 2015 -0400
    add new tokens

commit 832584b3f8020240fd6f591752b3f3257bdaf58f
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Oct 25 18:59:51 2015 -0400
    Add success message if program is scanned and parsed successfully

commit 1b8cc34377e6a3f8566aa7beaf35109d80142188
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Oct 25 18:57:26 2015 -0400
    Implement single-line comments per the LRM

commit 42e0b295c2ed030bc293e5bf291ef6056d5feaed
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Sun Oct 25 18:48:47 2015 -0400
```

```
    Build a skeleton parser for flow-lang

commit eec84db53ac279f0f2edf4c6c713858e8a7ffa1f
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Oct 21 17:22:49 2015 -0400
    Rename the microc program to flowc for "flow compiler"

commit a721734a2e3cae677081ecc0a533e652c417e0e9
Author: Adam Chelminski <chelminski.adam@gmail.com>
Date:   Wed Oct 21 17:11:13 2015 -0400
    Add microc compiler code as skeleton code

commit 7ae3a6713af9b891f53b8b5b43a322ccf2ce19ae
Author: Mitchell Gouzenko <mgouzenko@gmail.com>
Date:   Sun Sep 13 16:17:59 2015 -0400
    Initial commit
```

5 Architectural Design

5.1 Diagram of Flow Compilation Process



5.2 Scanner

Relevant source code (in Appendix): `scanner.mll`

The scanner uses ocamllex to translate the Flow source code into a stream of tokens.

5.3 Parser

Relevant source code (in Appendix): `parser.mly, ast.ml`

The parser uses ocamlyacc to build an abstract syntax tree from the stream of tokens generated by the scanner. The rules for the abstract syntax tree are defined in `ast.ml`. Source code that makes it through the parser and successfully transformed into an abstract syntax tree is syntactically valid.

5.4 Semantic Analyzer

Relevant source code (in Appendix): `semantic_analysis.ml, sast.ml`

In semantic analysis, we take the abstract syntax tree and annotate it, producing a semantically checked abstract syntax tree (SAST). There is more or less a one-to-one mapping between components of the AST and SAST. The primary difference is that the SAST only contains expressions whose types have been checked. Semantic analysis performs quite a few checks that are unique to Flow. The most important ones include:

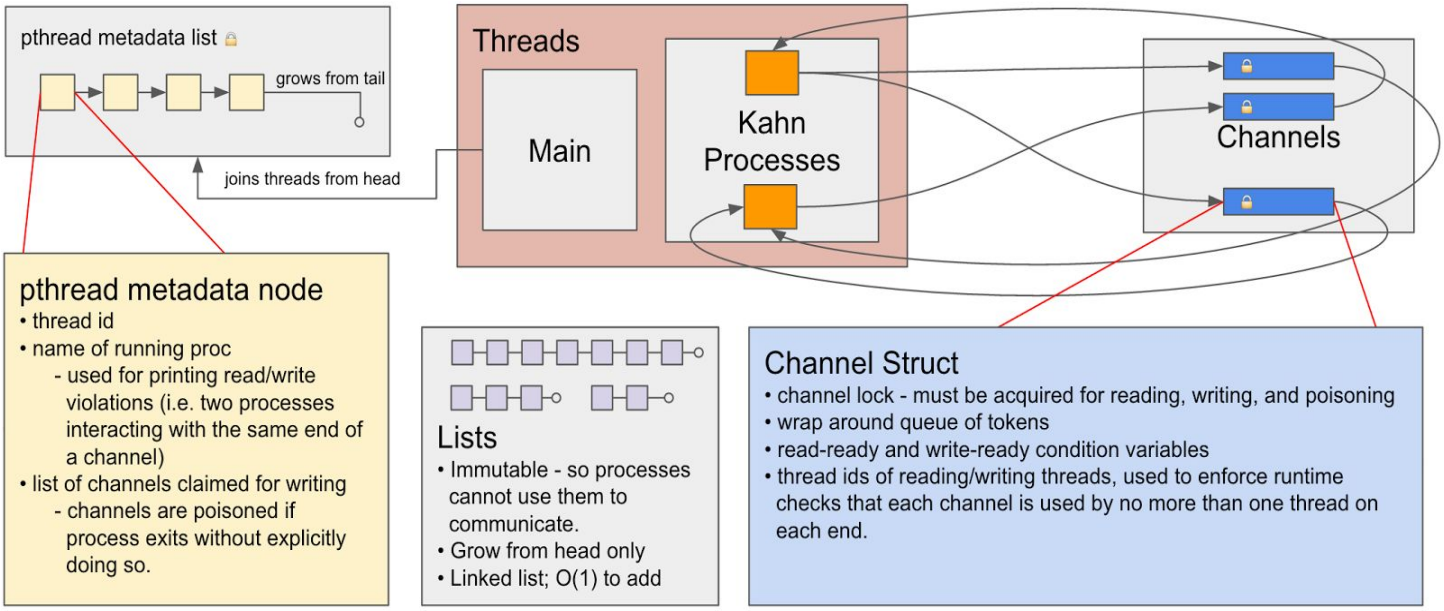
- Enforcing the rule that global variables cannot be assigned to
- Ensuring that only `in` channels are read from and `out` channels are written to
- Checking types in read/write operations on channels
- Ensuring that only `out` channels are ever poisoned
- Ensuring that channels can be used in a boolean context

5.5 Compiler

Relevant source code (in Appendix): `compile.ml`

The compiler converts the semantically checked abstract syntax tree (sast) that results from semantic analysis into C code. The C code generation is done by translating each component of the sast into the appropriate C constructs. The generated C code relies heavily on the C runtime environment described below.

5.6 The Runtime Environment



Relevant source code (in Appendix): `c_runtime.c`

5.6.1 The pthread metadata list (implemented by Mitchell)

The c runtime environment's job is to manage the interactions between processes and channels. All processes are started on separate threads, and those threads are joined in main. There's a global join list - henceforth termed the "pthread metadata list" - that contains the ids of all pthreads that are running. The pthread metadata list is protected by a global lock, and implemented as a linked list of `struct _pthread_node`.

When main invokes processes, it adds their ids to this list. Then, after the body of main has finished executing, the function `_wait_for_finish` is called. This function attempts to join all threads starting from the head of the list. If a process invokes another process, it adds that process's entry to the end of the list and adjusts the tail. That way, the main thread will never overlook processes that are added to the list.

To see why, consider a hypothetical process A, which tries to start process B. The main thread may be waiting on process A to halt. Before A halts though, it adds B's entry to the thread list. By the time the main thread joins A, the entry for B will already be in the list. Thus, as the main thread advances through the thread list, it's guaranteed to see B's entry, even if A had previously been the last unjoined thread.

A process's entry in the thread metadata list also contains a linked list of channels that the process has written to. When the process exits, if those channels have not been poisoned, the runtime poisons them automatically in the function `_exit_thread`.

5.6.2 Channels (implemented by Mitchell and Hyonjee)

There are several types of channels in the flow environment. In fact, there are channel structures associated with each supported token type. These structures all share common members defined by the macro `BASIC_CHANNEL_MEMBERS`.

Member	Function
`pthread_mutex_t lock;`	Lock that must be acquired to modify channel
`int size;`	Current size of the channel
`bool poisoned;`	Whether or not the channel has been poisoned
`pthread_cond_t write_ready;`	Condition variable signifying that the channel has space in it for more tokens to be written
`pthread_cond_t read_ready;`	Condition variable signifying that the channel has tokens to read
`int front;`	The index of the front of the queue
`int back;`	The index of the back of the queue
`bool claimed_for_writing;`	Whether or not a process has written to the channel and claimed it for writing
`bool claimed_for_reading;`	Whether or not a process has read from the channel and claimed it for reading
`pthread_t writing_thread;`	The thread id of the single process allowed to write to this channel
`pthread_t reading_thread;`	The thread id of the single process allowed to read this channel

Each channel is implemented as a producer consumer wrap-around queue with a fixed size of 100 elements. For each channel structure, there are functions to enqueue and dequeue tokens. Because the enqueue and dequeue operations are similar for all channels, these functions are generated by the macros `MAKE_ENQUEUE_FUNC(type)` and `MAKE_DEQUEUE_FUNC(type)`.

5.6.2.1 Channel Reads and Writes - Enqueueing and Dequeueing

When a process attempts to issue a read or write on a channel, its identity must first be established. If a process is attempting to read from a channel, its thread id is checked against that channel's `reading_thread` member. If the two don't match, a runtime error occurs.

If a channel has never been read from, its `claimed_for_reading` flag is initially false. When the channel is read from for the first time, this flag is set to true, and `reading_thread` is set to the thread's id.

The same procedure is used for writing to channels. In addition, when a process writes to a channel for the first time, the channel is added to that process's node in the thread metadata list.

5.6.2.2 Channels in a Boolean Context

As previously discussed, Flow allows the programmer to query the status of a channel by putting that channel in a conditional statement. This behavior is implemented by the function `wait_for_more(struct _channel *channel)`.

5.6.3 Lists (implemented by Mitchell and Hyonjee)

Flow lists are immutable and implemented as linked lists. Lists are only allowed to grow from the head. When a new element is added to a list, it's put in a `struct _cell` and set to point at the old list head. The pointer to the new cell is then returned. Immutability of lists means that multiple lists can share the same tail.

5.6.4 Dot Graph Feature (implemented by Mitchell)

When the flow compiler is invoked with the "-d" option, it compiles to c code that generates a dot graph when it is run. This dot graph is output through stderr, and can be collected and compiled into a visual representation of the KPN made by the program.

6 Test Plan

The shell script testall.sh runs our automated test suite which has a total of 80 tests. Test names that begin with "fail-" should check for both compile time and runtime errors. We found it important to check for runtime errors since our runtime has assertions that prevent actions such as writing to a poisoned channel. Originally, we had our "fail-" tests check that the error message matched a corresponding out file, but found this unproductive since we regularly improved our error messages, which meant that all the out files would have to be updated. Test names that begin with "test-" should check that the output successfully matches the corresponding .out file. In this scenario, tests are required to print to stdout. We found this to be the better than matching the compiler output to C code since making changes in the compiler would require a rewrite of all the tests. Both sets of tests print the test name and "OK" when passing, and print large failure messages when a test is failing. Tests that should match an output should will show the comparison and the lines that differ. Running the test script with the flag -k will keep all intermediate files such as the generated C code.

Tests were developed with a couple different mindsets and strategies. The first strategy was to write tests as the compiler evolved, ensuring that key components were working as they were being implemented. The second strategy, which took place later in the project evolution, was to think deviously, intentionally writing tests that would break the Flow language.

Included among the tests are two implementations of a bitonic sorter (a parallel sorting network). One is a manually built one that can sort inputs of size 4, and one is a recursively defined one that can sort inputs of size 2^N for some arbitrary natural number N. These tests are named test-bitonic-manual and test-bitonic-recursive, respectively.

6.1 Test Suite

6.1.1 testall.sh (implemented by Zach)

```
#!/bin/sh

FLOWC="./flowc"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.flow files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
  echo "FAILED"
  error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
      SignalError "$1 differs"
      echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
```

```
      SignalError "$1 failed on $*"
      return 1
    }
}

RunFail() {
    # echo $* 1>&2
    eval $* || {
      return 1
    }
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.flow//'`
    reffile=`echo $1 | sed 's/.flow$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.c" &&
    Run "$FLOWC" "-c" $1 ">" ${basename}.c &&
    gcc ${basename}.c  &&
    ./a.out > ${basename}.c.out
    Compare ${basename}.c.out ${reffile}.out ${basename}.c.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.flow//'`
    reffile=`echo $1 | sed 's/.flow$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    if generatedfiles="$generatedfiles ${basename}.c" && RunFail "$FLOWC" "-c" $1 "&>" ${basename}.c; then
      error=1

      # Check for runtime error
      generatedfiles="$generatedfiles ${basename}.c" &&
      Run "$FLOWC" "-c" $1 ">" ${basename}.c &&
      gcc ${basename}.c  &&
```

```
      if ./a.out;then
        error=1;
      else
        error=0;
      fi

    fi
    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
        rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "############ FAILED ###############"
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
}

make >> $globallog

while getopts kdpsh c; do
    case $c in
  k) # Keep intermediate files
      keep=1
      ;;
  h) # Help
      Usage
      ;;
    esac
done

shift `expr $OPTIND - 1`

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/fail-*.flow tests/test-*.flow"
fi

for file in $files
do
    case $file in
  *test-*)
      Check $file 2>> $globallog
      ;;
  *fail-*)
      CheckFail $file 2>> $globallog
      ;;
  *)
      echo "unknown file type $file"
      globalerror=1
      ;;
    esac
done

if [ $keep -eq 0 ] ; then
  make clean >> $globallog
fi

exit $globalerror
```

6.1.2 Test Suite Output

```
-n fail-access-empty-list1...
OK
-n fail-access-empty-list2...
OK
-n fail-arith1...
OK
-n fail-bad-type-to-chan...
OK
-n fail-break...
OK
-n fail-chan-of-chan...
OK
-n fail-chan-of-lists...
OK
-n fail-continue...
OK
-n fail-decl1...
OK
-n fail-func-decl-without-init...
OK
-n fail-func-undecl...
OK
-n fail-immutable-global...
OK
-n fail-list-init1...
OK
-n fail-list-init2...
OK
-n fail-list-init3...
OK
-n fail-list-init4...
OK
-n fail-no-return...
OK
-n fail-poison...
OK
-n fail-poison2...
OK
-n fail-read-from-out-chan...
OK
-n fail-single-in-for-chan...
OK
-n fail-single-out-for-chan...
OK
-n fail-write-to-in-chan...
OK
-n fail-write-to-nodir-chan...
OK
-n test-arith1...
OK
-n test-arith2...
OK
-n test-assoc-concat...
OK
-n test-assoc-equal...
OK
-n test-assoc-negate...
OK
-n test-assoc-tail...
OK
-n test-bitonic-manual...
OK
```

```
-n test-bitonic-recursive...
OK
-n test-break...
OK
-n test-chan-return...
OK
-n test-chan-return2...
OK
-n test-continue...
OK
-n test-empty-func-decl...
OK
-n test-fib...
OK
-n test-for1...
OK
-n test-func-no-args...
OK
-n test-func-no-body...
OK
-n test-func-with-chan...
OK
-n test-func-with-chan2...
OK
-n test-func1...
OK
-n test-if1...
OK
-n test-if2...
OK
-n test-list-add-front...
OK
-n test-list-channel...
OK
-n test-list-char...
OK
-n test-list-double...
OK
-n test-list-empty-init...
OK
-n test-list-empty-tail...
OK
-n test-list-immutable...
OK
-n test-list-init...
OK
-n test-list-init2...
OK
-n test-list-init3...
OK
-n test-list-length...
OK
-n test-list-return...
OK
-n test-list-return2...
OK
-n test-mutable...
OK
-n test-pass-list-to-function...
OK
-n test-pass-list-to-function2...
OK
-n test-print...
OK
-n test-proc-in-proc...
OK
-n test-proc-no-poison...
```

```
OK
-n test-rand...
OK
-n test-recursion...
OK
-n test-return-process...
OK
-n test-running-sum...
OK
-n test-scope1...
OK
-n test-scope2...
OK
-n test-scope3...
OK
-n test-scope4...
OK
-n test-scope5...
OK
-n test-simple-interleaver-char...
OK
-n test-simple-interleaver-double...
OK
-n test-simple-interleaver...
OK
-n test-string1...
OK
-n test-sum...
OK
-n test-while1...
OK
```

6.2 Flow to C code Generation

6.2.1 sum.flow

```
proc numGen(out int ochan){
  list <int> test = [1, 2, 3, 4, 5];

  while(#test > 0) {
    @test -> ochan;
    test = ^test;
  }

  poison ochan;
}

proc sum(in int chan) {
  int sum = 0;
  while(chan) {
    sum = sum + @chan;
  }
  print_int(sum);
}

int main() {
  channel<int> chan;
  numGen(chan);
  sum(chan);
}
```

6.2.2 sum.c (formatted with clang-format)

```
#include <assert.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

/***************************** Channel Structs ******************************/
#define BASIC_CHANNEL_MEMBERS                                                  \
  pthread_mutex_t lock;                                                        \
  int size;                                                                    \
  bool poisoned;                                                               \
  pthread_cond_t write_ready;                                                  \
  pthread_cond_t read_ready;                                                   \
  int front;                                                                   \
  int back;                                                                    \
  int MAX_SIZE;                                                                \
  int claimed_for_writing;                                                     \
  int claimed_for_reading;                                                     \
  pthread_t writing_thread;                                                    \
  pthread_t reading_thread;

struct _channel {
  BASIC_CHANNEL_MEMBERS
};

struct _int_channel {
  BASIC_CHANNEL_MEMBERS
  int queue[100];
};

struct _char_channel {
  BASIC_CHANNEL_MEMBERS
  char queue[100];
};

struct _double_channel {
  BASIC_CHANNEL_MEMBERS
  double queue[100];
};

#define MALLOC_CHANNEL(type)                                                   \
  = (struct _##type##_channel *)malloc(sizeof(struct _##type##_channel));

int _init_channel(struct _channel *channel) {
  if (pthread_mutex_init(&channel->lock, NULL) != 0) {
    printf("Mutex init failed");
    return 1;
  }

  if (pthread_cond_init(&channel->write_ready, NULL) +
          pthread_cond_init(&channel->read_ready, NULL) !=
      0) {
    printf("Cond init failed");
    return 1;
  }
  channel->claimed_for_reading = 0;
  channel->claimed_for_writing = 0;
```

```
  channel->MAX_SIZE = 100;
  channel->front = 0;
  channel->back = 0;
  channel->poisoned = false;
  return 0;
}

/************************ Global Thread Metadata *************************/

/* Node for linked list of channel names. Keeps track of channels that
 * threads can write to. */
struct _channel_list_node {
  struct _channel *chan;
  struct _channel_list_node *next;
};

/* Defines a node of the global thread metadata list. */
struct _pthread_node {
  pthread_t thread;
  struct _pthread_node *next;
  char *proc_name;
  struct _channel_list_node *writing_channels;
};

/* The global thread metadata list */
struct _pthread_node *_head = NULL;
struct _pthread_node *_tail = NULL;

/* Lock for the thread metadata list */
pthread_mutex_t _thread_list_lock;
pthread_mutex_t _ref_counting_lock;

/* Finds a thread in the global threadlist given its id */
struct _pthread_node *_get_thread(pthread_t thread_id) {
  pthread_mutex_lock(&_thread_list_lock);
  struct _pthread_node *curr = _head;
  while (curr) {
    if (curr->thread == thread_id) {
      break;
    }
    curr = curr->next;
  }
  pthread_mutex_unlock(&_thread_list_lock);
  return curr;
}

/* Gets the name of the process running on a thread,
 * given the thread id */
char *_get_thread_name(pthread_t thread_id) {
  if (_head == NULL)
    return "";
  pthread_mutex_lock(&_thread_list_lock);
  char *name = "";
  struct _pthread_node *curr = _head;
  while (curr) {
    if (curr->thread == thread_id) {
      name = curr->proc_name;
      break;
    }
    curr = curr->next;
  }
  pthread_mutex_unlock(&_thread_list_lock);
  return name;
}

void _print_dot_node(struct _channel *chan) {
  fprintf(stderr, "{%d[label=%s]}->{%d[label=%s]}\n", (int)chan->writing_thread,
```

```
          _get_thread_name(chan->writing_thread), (int)chan->reading_thread,
          _get_thread_name(chan->reading_thread));
}

/************************ Enqueue/Dequeue Macros ****************************/

/* Given a token type, this macro generates an enqueue function
 * for the associated channel. */
#define MAKE_ENQUEUE_FUNC(type)                                              \
  type _enqueue_##type(type element, struct _##type##_channel *channel,      \
                       bool dot_print) {                                     \
    pthread_mutex_lock(&channel->lock);                                      \
    pthread_t this_thread = pthread_self();                                  \
    if (!channel->claimed_for_writing) {                                     \
      channel->claimed_for_writing = 1;                                      \
      channel->writing_thread = this_thread;                                 \
      struct _pthread_node *this_thread_node = _get_thread(this_thread);     \
      struct _channel_list_node *new_writing_chan =                          \
          malloc(sizeof(struct _channel_list_node));                         \
      new_writing_chan->next = this_thread_node->writing_channels;           \
      new_writing_chan->chan = (struct _channel *)channel;                   \
      this_thread_node->writing_channels = new_writing_chan;                 \
      if (channel->claimed_for_reading && dot_print)                         \
        _print_dot_node((struct _channel *)channel);                         \
    } else if (channel->writing_thread != this_thread) {                     \
      fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "   \
                      "write to a channel belonging to %s (thread 0x%x)\n",  \
              _get_thread_name(this_thread), (int)this_thread,               \
              _get_thread_name(channel->writing_thread),                     \
              (int)channel->writing_thread);                                 \
      exit(1);                                                               \
    }                                                                        \
    while (channel->size >= channel->MAX_SIZE)                               \
      pthread_cond_wait(&channel->write_ready, &channel->lock);              \
    assert(channel->size < channel->MAX_SIZE);                               \
    if (channel->poisoned) {                                                 \
      fprintf(stderr,                                                        \
              "Attempting to read from a channel that is empty and poisoned"); \
      exit(1);                                                               \
    }                                                                        \
    channel->queue[channel->back] = element;                                 \
    channel->back = (channel->back + 1) % channel->MAX_SIZE;                 \
    channel->size++;                                                         \
    pthread_cond_signal(&channel->read_ready);                               \
    pthread_mutex_unlock(&channel->lock);                                    \
    return element;                                                          \
  }

/* Create enqueue functions for ints, chars, and doubles */
MAKE_ENQUEUE_FUNC(int)
MAKE_ENQUEUE_FUNC(char)
MAKE_ENQUEUE_FUNC(double)

/* This macro calls the appropriate dequeue function */
#define CALL_ENQUEUE_FUNC(e, c, t, dot) _enqueue_##t(e, c, dot)

/* Given a token type, this macro generates a dequeue function
 * for the associated channel. */
#define MAKE_DEQUEUE_FUNC(type)                                              \
  type _dequeue_##type(struct _##type##_channel *channel, bool dot_print) {  \
    pthread_mutex_lock(&channel->lock);                                      \
    pthread_t this_thread = pthread_self();                                  \
    if (!channel->claimed_for_reading) {                                     \
      channel->claimed_for_reading = 1;                                      \
      channel->reading_thread = this_thread;                                 \
      if (channel->claimed_for_writing && dot_print)                         \
```

```
        _print_dot_node((struct _channel *)channel);                          \
    } else if (channel->reading_thread != this_thread) {                      \
      fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "    \
                      "read from a channel belonging to %s (thread 0x%x)\n",  \
              _get_thread_name(this_thread), (int)this_thread,                \
              _get_thread_name(channel->reading_thread),                      \
              (int)channel->reading_thread);                                  \
      exit(1);                                                                \
    }                                                                         \
    if (channel->size == 0) {                                                 \
      fprintf(stderr, "Attempting to read from empty channel");               \
      exit(1);                                                                \
    }                                                                         \
    type result = channel->queue[channel->front];                             \
    channel->front = (channel->front + 1) % channel->MAX_SIZE;                \
    channel->size--;                                                          \
    pthread_cond_signal(&channel->write_ready);                               \
    pthread_mutex_unlock(&channel->lock);                                     \
    return result;                                                            \
  }

/* Make dequeue functions for int, char, and double channels */
MAKE_DEQUEUE_FUNC(int)
MAKE_DEQUEUE_FUNC(char)
MAKE_DEQUEUE_FUNC(double)

/* This macro calls the appropriate dequeue function */
#define CALL_DEQUEUE_FUNC(c, t, dot) _dequeue_##t(c, dot)

/* Poison the channel, indicating that it won't be written to in the future */
void _poison(struct _channel *channel) {
  pthread_mutex_lock(&channel->lock);
  channel->poisoned = true;
  pthread_cond_signal(&channel->read_ready);
  pthread_mutex_unlock(&channel->lock);
}

/* This function is called whenever a channel is used in a boolean context.
 * Three cases:
 *   1) Channel is poisoned and empty     -> return false
 *   2) Channel is nonempty               -> return true
 *   3) Channel is empty but not poisoned -> block
 */
bool _wait_for_more(struct _channel *channel) {
  pthread_mutex_lock(&channel->lock);
  while (channel->size == 0) {
    if (channel->poisoned) {
      pthread_mutex_unlock(&channel->lock);
      return false;
    } else {
      pthread_cond_wait(&channel->read_ready, &channel->lock);
    }
  }
  pthread_mutex_unlock(&channel->lock);
  return true;
}

/************************ Miscellaneous ************************/

/* Initializes global locks */
void _initialize_runtime(bool print_dot) {
  pthread_mutex_init(&_thread_list_lock, NULL);
  pthread_mutex_init(&_ref_counting_lock, NULL);
  srand(time(NULL));
  if (print_dot)
    fprintf(stderr, "digraph G{\n");
}
```

```
/* Create a pthread node and enqueue it on the list. Return the address of
 * its id for pthread_create */
pthread_t *_make_pthread_t(char *proc_name) {
  pthread_mutex_lock(&_thread_list_lock);
  struct _pthread_node *new_pthread =
      (struct _pthread_node *)malloc(sizeof(struct _pthread_node));
  new_pthread->next = NULL;
  new_pthread->proc_name = proc_name;
  new_pthread->writing_channels = NULL;
  if (_head == NULL) {
    _head = _tail = new_pthread;
  } else {
    _tail->next = new_pthread;
    _tail = new_pthread;
  }
  pthread_mutex_unlock(&_thread_list_lock);
  return &(new_pthread->thread);
}

/* Invoked when return is reached from a process.
 * This function will cause the returning thread to poison all of
 * its outgoing channels if it hasn't done so yet. */
void _exit_thread() {
  struct _pthread_node *this_thread = _get_thread(pthread_self());
  struct _channel_list_node *curr_chan = this_thread->writing_channels;
  while (curr_chan) {
    if (!curr_chan->chan->poisoned)
      _poison(curr_chan->chan);
    curr_chan = curr_chan->next;
  }
  pthread_exit(NULL);
}

/* Called from within main to wait for processes to finish */
void _wait_for_finish(bool print_dot) {
  struct _pthread_node *curr = _head;
  while (curr) {
    pthread_join(curr->thread, NULL);
    curr = curr->next;
  }
  if (print_dot)
    fprintf(stderr, "}");
}

/****************************** Lists *******************************/
union _payload {
  int _int;
  double _double;
  char _char;
  void *_cell;
  struct _int_channel *_int_channel;
};

struct _cell {
  struct _cell *next;
  union _payload data;
  int references;
  int length;
};

struct _cell *_add_front(union _payload element, struct _cell *tail) {
  struct _cell *new_cell = malloc(sizeof(struct _cell));
  new_cell->references = 1;
  new_cell->data = element;
  new_cell->next = tail;
```

```
  if (!tail)
    new_cell->length = 1;
  else {
    new_cell->length = tail->length + 1;
    tail->references++;
  }
  return new_cell;
}

struct _cell *_get_tail(struct _cell *head) {
  if (!head) {
    fprintf(stderr, "Runtime error: cannot get tail of empty list");
    exit(1);
  }

  return head->next;
}

void __decrease_refs(struct _cell *head, int lock) {
  if (lock)
    pthread_mutex_lock(&_ref_counting_lock);

  if (!head) {
    if (lock)
      pthread_mutex_unlock(&_ref_counting_lock);
    return;
  } else if (head->references > 1)
    head->references--;
  else {
    __decrease_refs(head->next, 0);
    free(head);
  }
  if (lock)
    pthread_mutex_unlock(&_ref_counting_lock);
}

void _decrease_refs(struct _cell *head) {
  //__decrease_refs(head, 1);
}

void _increase_refs(struct _cell *head) {
  pthread_mutex_lock(&_ref_counting_lock);
  if (head)
    head->references++;
  pthread_mutex_unlock(&_ref_counting_lock);
}

union _payload _get_front(struct _cell *head) {
  if (!head) {
    fprintf(stderr, "Runtime error: cannot get head of empty list");
    exit(1);
  }

  return head->data;
}

int _get_length(struct _cell *head) {
  if (!head)
    return 0;
  return head->length;
}
struct _numGen_args {
  struct _int_channel *ochan;
};
void *numGen(void *_args) {
  struct _int_channel *ochan = ((struct _numGen_args *)_args)->ochan;
```

```
  struct _cell *temp;
  struct _cell *test = NULL;
  test = _add_front((union _payload)5, test);
  test = _add_front((union _payload)4, test);
  test = _add_front((union _payload)3, test);
  test = _add_front((union _payload)2, test);
  test = _add_front((union _payload)1, test);
  while (_get_length(test) > 0) {
    CALL_ENQUEUE_FUNC(_get_front(test)._int, ochan, int, false);
    temp = test;
    test = _get_tail(test);
    _increase_refs(test);
    _decrease_refs(temp);
  }
  _poison((struct _channel *)ochan);
  _exit_thread();
}
struct _sum_args {
  struct _int_channel *chan;
};
void *sum(void *_args) {
  struct _int_channel *chan = ((struct _sum_args *)_args)->chan;
  struct _cell *temp;
  int sum = 0;
  while (_wait_for_more((struct _channel *)chan)) {
    sum = sum + CALL_DEQUEUE_FUNC(chan, int, false);
  }
  printf("%d", sum);
  fflush(stdout);
  _exit_thread();
}
int main() {
  _initialize_runtime(false);
  struct _cell *temp;
  struct _int_channel *chan MALLOC_CHANNEL(int);
  _init_channel((struct _channel *)chan);
  {
    pthread_t *_t = _make_pthread_t("numGen");
    struct _numGen_args *_margs = malloc(sizeof(struct _numGen_args));
    struct _numGen_args _args = {chan};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, numGen, (void *)_margs);
  };
  {
    pthread_t *_t = _make_pthread_t("sum");
    struct _sum_args *_margs = malloc(sizeof(struct _sum_args));
    struct _sum_args _args = {chan};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, sum, (void *)_margs);
  };
  _wait_for_finish(false);
}
```

6.2.3 int_interleaver.flow

```
proc tokenGen(out int ochan, int token) {
  token -> ochan;
  poison ochan;
}

proc printer(in int chan) {
  while(chan) {
    print_int(@chan);
    println();
```

```
  }
}

proc interleaver(in int chan1, in int chan2, out int ochan) {
  while(chan1 || chan2){
    if(chan1) {@chan1 -> ochan;}
    if(chan2) {@chan2 -> ochan;}
  }
  poison ochan;
}

int main() {
  channel<int> chan1;
  channel<int> chan2;
  channel<int> chan3;
  int int1 = 1;
  int int2 = 2;
  tokenGen(chan1, int1);
  tokenGen(chan2, int2);
  interleaver(chan1, chan2, chan3);
  printer(chan3);
}
```

6.2.4 int_interleaver.c

```
/* c_runtime.c */
#include <assert.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

/****************************** Channel Structs ******************************/
#define BASIC_CHANNEL_MEMBERS                                                  \
  pthread_mutex_t lock;                                                        \
  int size;                                                                    \
  bool poisoned;                                                               \
  pthread_cond_t write_ready;                                                  \
  pthread_cond_t read_ready;                                                   \
  int front;                                                                   \
  int back;                                                                    \
  int MAX_SIZE;                                                                \
  int claimed_for_writing;                                                     \
  int claimed_for_reading;                                                     \
  pthread_t writing_thread;                                                    \
  pthread_t reading_thread;

struct _channel {
  BASIC_CHANNEL_MEMBERS
};

struct _int_channel {
  BASIC_CHANNEL_MEMBERS
  int queue[100];
};

struct _char_channel {
  BASIC_CHANNEL_MEMBERS
  char queue[100];
};
```

```
struct _double_channel {
  BASIC_CHANNEL_MEMBERS
  double queue[100];
};

#define MALLOC_CHANNEL(type)                                                \
  = (struct _##type##_channel *)malloc(sizeof(struct _##type##_channel));

int _init_channel(struct _channel *channel) {
  if (pthread_mutex_init(&channel->lock, NULL) != 0) {
    printf("Mutex init failed");
    return 1;
  }

  if (pthread_cond_init(&channel->write_ready, NULL) +
        pthread_cond_init(&channel->read_ready, NULL) !=
      0) {
    printf("Cond init failed");
    return 1;
  }
  channel->claimed_for_reading = 0;
  channel->claimed_for_writing = 0;
  channel->MAX_SIZE = 100;
  channel->front = 0;
  channel->back = 0;
  channel->poisoned = false;
  return 0;
}

/************************ Global Thread Metadata *************************/

/* Node for linked list of channel names. Keeps track of channels that
 * threads can write to. */
struct _channel_list_node {
  struct _channel *chan;
  struct _channel_list_node *next;
};

/* Defines a node of the global thread metadata list. */
struct _pthread_node {
  pthread_t thread;
  struct _pthread_node *next;
  char *proc_name;
  struct _channel_list_node *writing_channels;
};

/* The global thread metadata list */
struct _pthread_node *_head = NULL;
struct _pthread_node *_tail = NULL;

/* Lock for the thread metadata list */
pthread_mutex_t _thread_list_lock;
pthread_mutex_t _ref_counting_lock;

/* Finds a thread in the global threadlist given its id */
struct _pthread_node *_get_thread(pthread_t thread_id) {
  pthread_mutex_lock(&_thread_list_lock);
  struct _pthread_node *curr = _head;
  while (curr) {
    if (curr->thread == thread_id) {
      break;
    }
    curr = curr->next;
  }
  pthread_mutex_unlock(&_thread_list_lock);
```

```
  return curr;
}

/* Gets the name of the process running on a thread,
 * given the thread id */
char *_get_thread_name(pthread_t thread_id) {
  if (_head == NULL)
    return "";
  pthread_mutex_lock(&_thread_list_lock);
  char *name = "";
  struct _pthread_node *curr = _head;
  while (curr) {
    if (curr->thread == thread_id) {
      name = curr->proc_name;
      break;
    }
    curr = curr->next;
  }
  pthread_mutex_unlock(&_thread_list_lock);
  return name;
}

void _print_dot_node(struct _channel *chan) {
  fprintf(stderr, "{%d[label=%s]}->{%d[label=%s]}\n", (int)chan->writing_thread,
          _get_thread_name(chan->writing_thread), (int)chan->reading_thread,
          _get_thread_name(chan->reading_thread));
}

/************************ Enqueue/Dequeue Macros ****************************/

/* Given a token type, this macro generates an enqueue function
 * for the associated channel. */
#define MAKE_ENQUEUE_FUNC(type)                                              \
  type _enqueue_##type(type element, struct _##type##_channel *channel,      \
                       bool dot_print) {                                     \
    pthread_mutex_lock(&channel->lock);                                      \
    pthread_t this_thread = pthread_self();                                  \
    if (!channel->claimed_for_writing) {                                     \
      channel->claimed_for_writing = 1;                                      \
      channel->writing_thread = this_thread;                                 \
      struct _pthread_node *this_thread_node = _get_thread(this_thread);     \
      struct _channel_list_node *new_writing_chan =                          \
          malloc(sizeof(struct _channel_list_node));                         \
      new_writing_chan->next = this_thread_node->writing_channels;           \
      new_writing_chan->chan = (struct _channel *)channel;                   \
      this_thread_node->writing_channels = new_writing_chan;                 \
      if (channel->claimed_for_reading && dot_print)                         \
        _print_dot_node((struct _channel *)channel);                         \
    } else if (channel->writing_thread != this_thread) {                     \
      fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "   \
                      "write to a channel belonging to %s (thread 0x%x)\n",  \
              _get_thread_name(this_thread), (int)this_thread,               \
              _get_thread_name(channel->writing_thread),                     \
              (int)channel->writing_thread);                                 \
      exit(1);                                                               \
    }                                                                        \
    while (channel->size >= channel->MAX_SIZE)                               \
      pthread_cond_wait(&channel->write_ready, &channel->lock);              \
    assert(channel->size < channel->MAX_SIZE);                               \
    if (channel->poisoned) {                                                 \
      fprintf(stderr,                                                        \
              "Attempting to read from a channel that is empty and poisoned"); \
      exit(1);                                                               \
    }                                                                        \
    channel->queue[channel->back] = element;                                 \
    channel->back = (channel->back + 1) % channel->MAX_SIZE;                 \
    channel->size++;                                                         \
```

```
    pthread_cond_signal(&channel->read_ready);                                  \
    pthread_mutex_unlock(&channel->lock);                                       \
    return element;                                                             \
  }

/* Create enqueue functions for ints, chars, and doubles */
MAKE_ENQUEUE_FUNC(int)
MAKE_ENQUEUE_FUNC(char)
MAKE_ENQUEUE_FUNC(double)

/* This macro calls the appropriate dequeue function */
#define CALL_ENQUEUE_FUNC(e, c, t, dot) _enqueue_##t(e, c, dot)

/* Given a token type, this macro generates a dequeue function
 * for the associated channel. */
#define MAKE_DEQUEUE_FUNC(type)                                                 \
  type _dequeue_##type(struct _##type##_channel *channel, bool dot_print) {     \
    pthread_mutex_lock(&channel->lock);                                         \
    pthread_t this_thread = pthread_self();                                     \
    if (!channel->claimed_for_reading) {                                        \
      channel->claimed_for_reading = 1;                                         \
      channel->reading_thread = this_thread;                                    \
      if (channel->claimed_for_writing && dot_print)                            \
        _print_dot_node((struct _channel *)channel);                            \
    } else if (channel->reading_thread != this_thread) {                        \
      fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "      \
                      "read from a channel belonging to %s (thread 0x%x)\n",    \
              _get_thread_name(this_thread), (int)this_thread,                  \
              _get_thread_name(channel->reading_thread),                        \
              (int)channel->reading_thread);                                    \
      exit(1);                                                                  \
    }                                                                           \
    if (channel->size == 0) {                                                   \
      fprintf(stderr, "Attempting to read from empty channel");                 \
      exit(1);                                                                  \
    }                                                                           \
    type result = channel->queue[channel->front];                               \
    channel->front = (channel->front + 1) % channel->MAX_SIZE;                  \
    channel->size--;                                                            \
    pthread_cond_signal(&channel->write_ready);                                 \
    pthread_mutex_unlock(&channel->lock);                                       \
    return result;                                                              \
  }

/* Make dequeue functions for int, char, and double channels */
MAKE_DEQUEUE_FUNC(int)
MAKE_DEQUEUE_FUNC(char)
MAKE_DEQUEUE_FUNC(double)

/* This macro calls the appropriate dequeue function */
#define CALL_DEQUEUE_FUNC(c, t, dot) _dequeue_##t(c, dot)

/* Poison the channel, indicating that it won't be written to in the future */
void _poison(struct _channel *channel) {
  pthread_mutex_lock(&channel->lock);
  channel->poisoned = true;
  pthread_cond_signal(&channel->read_ready);
  pthread_mutex_unlock(&channel->lock);
}

/* This function is called whenever a channel is used in a boolean context.
 * Three cases:
 *   1) Channel is poisoned and empty     -> return false
 *   2) Channel is nonempty               -> return true
 *   3) Channel is empty but not poisoned -> block
 */
```

```
bool _wait_for_more(struct _channel *channel) {
  pthread_mutex_lock(&channel->lock);
  while (channel->size == 0) {
    if (channel->poisoned) {
      pthread_mutex_unlock(&channel->lock);
      return false;
    } else {
      pthread_cond_wait(&channel->read_ready, &channel->lock);
    }
  }
  pthread_mutex_unlock(&channel->lock);
  return true;
}

/************************ Miscellaneous ************************/

/* Initializes global locks */
void _initialize_runtime(bool print_dot) {
  pthread_mutex_init(&_thread_list_lock, NULL);
  pthread_mutex_init(&_ref_counting_lock, NULL);
  srand(time(NULL));
  if (print_dot)
    fprintf(stderr, "digraph G{\n");
}

/* Create a pthread node and enqueue it on the list. Return the address of
 * its id for pthread_create */
pthread_t *_make_pthread_t(char *proc_name) {
  pthread_mutex_lock(&_thread_list_lock);
  struct _pthread_node *new_pthread =
      (struct _pthread_node *)malloc(sizeof(struct _pthread_node));
  new_pthread->next = NULL;
  new_pthread->proc_name = proc_name;
  new_pthread->writing_channels = NULL;
  if (_head == NULL) {
    _head = _tail = new_pthread;
  } else {
    _tail->next = new_pthread;
    _tail = new_pthread;
  }
  pthread_mutex_unlock(&_thread_list_lock);
  return &(new_pthread->thread);
}

/* Invoked when return is reached from a process.
 * This function will cause the returning thread to poison all of
 * its outgoing channels if it hasn't done so yet. */
void _exit_thread() {
  struct _pthread_node *this_thread = _get_thread(pthread_self());
  struct _channel_list_node *curr_chan = this_thread->writing_channels;
  while (curr_chan) {
    if (!curr_chan->chan->poisoned)
      _poison(curr_chan->chan);
    curr_chan = curr_chan->next;
  }
  pthread_exit(NULL);
}

/* Called from within main to wait for processes to finish */
void _wait_for_finish(bool print_dot) {
  struct _pthread_node *curr = _head;
  while (curr) {
    pthread_join(curr->thread, NULL);
    curr = curr->next;
  }
  if (print_dot)
    fprintf(stderr, "}");
```

```
}

/****************************** Lists *******************************/
union _payload {
  int _int;
  double _double;
  char _char;
  void *_cell;
  struct _int_channel *_int_channel;
};

struct _cell {
  struct _cell *next;
  union _payload data;
  int references;
  int length;
};

struct _cell *_add_front(union _payload element, struct _cell *tail) {
  struct _cell *new_cell = malloc(sizeof(struct _cell));
  new_cell->references = 1;
  new_cell->data = element;
  new_cell->next = tail;
  if (!tail)
    new_cell->length = 1;
  else {
    new_cell->length = tail->length + 1;
    tail->references++;
  }
  return new_cell;
}

struct _cell *_get_tail(struct _cell *head) {
  if (!head) {
    fprintf(stderr, "Runtime error: cannot get tail of empty list");
    exit(1);
  }

  return head->next;
}

void __decrease_refs(struct _cell *head, int lock) {
  if (lock)
    pthread_mutex_lock(&_ref_counting_lock);

  if (!head) {
    if (lock)
      pthread_mutex_unlock(&_ref_counting_lock);
    return;
  } else if (head->references > 1)
    head->references--;
  else {
    __decrease_refs(head->next, 0);
    free(head);
  }
  if (lock)
    pthread_mutex_unlock(&_ref_counting_lock);
}

void _decrease_refs(struct _cell *head) {
  //__decrease_refs(head, 1);
}

void _increase_refs(struct _cell *head) {
  pthread_mutex_lock(&_ref_counting_lock);
  if (head)
```

```
    head->references++;
  pthread_mutex_unlock(&_ref_counting_lock);
}

union _payload _get_front(struct _cell *head) {
  if (!head) {
    fprintf(stderr, "Runtime error: cannot get head of empty list");
    exit(1);
  }

  return head->data;
}

int _get_length(struct _cell *head) {
  if (!head)
    return 0;
  return head->length;
}
struct _tokenGen_args {
  struct _int_channel *ochan;
  int token;
};
void *tokenGen(void *_args) {
  struct _int_channel *ochan = ((struct _tokenGen_args *)_args)->ochan;
  int token = ((struct _tokenGen_args *)_args)->token;
  struct _cell *temp;
  CALL_ENQUEUE_FUNC(token, ochan, int, false);
  _poison((struct _channel *)ochan);
  _exit_thread();
}
struct _printer_args {
  struct _int_channel *chan;
};
void *printer(void *_args) {
  struct _int_channel *chan = ((struct _printer_args *)_args)->chan;
  struct _cell *temp;
  while (_wait_for_more((struct _channel *)chan)) {
    printf("%d", CALL_DEQUEUE_FUNC(chan, int, false));
    fflush(stdout);
    printf("\n");
    fflush(stdout);
  }
  _exit_thread();
}
struct _interleaver_args {
  struct _int_channel *chan1;
  struct _int_channel *chan2;
  struct _int_channel *ochan;
};
void *interleaver(void *_args) {
  struct _int_channel *chan1 = ((struct _interleaver_args *)_args)->chan1;
  struct _int_channel *chan2 = ((struct _interleaver_args *)_args)->chan2;
  struct _int_channel *ochan = ((struct _interleaver_args *)_args)->ochan;
  struct _cell *temp;
  while (_wait_for_more((struct _channel *)chan1) ||
         _wait_for_more((struct _channel *)chan2)) {
    if (_wait_for_more((struct _channel *)chan1)) {
      CALL_ENQUEUE_FUNC(CALL_DEQUEUE_FUNC(chan1, int, false), ochan, int,
                        false);
    }

    else
      ;
    if (_wait_for_more((struct _channel *)chan2)) {
      CALL_ENQUEUE_FUNC(CALL_DEQUEUE_FUNC(chan2, int, false), ochan, int,
                        false);
    }
```

```
    else
      ;
  }
  _poison((struct _channel *)ochan);
  _exit_thread();
}
int main() {
  _initialize_runtime(false);
  struct _cell *temp;
  struct _int_channel *chan1 MALLOC_CHANNEL(int);
  _init_channel((struct _channel *)chan1);
  struct _int_channel *chan2 MALLOC_CHANNEL(int);
  _init_channel((struct _channel *)chan2);
  struct _int_channel *chan3 MALLOC_CHANNEL(int);
  _init_channel((struct _channel *)chan3);
  int int1 = 1;
  int int2 = 2;
  {
    pthread_t *_t = _make_pthread_t("tokenGen");
    struct _tokenGen_args *_margs = malloc(sizeof(struct _tokenGen_args));
    struct _tokenGen_args _args = {chan1, int1};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, tokenGen, (void *)_margs);
  };
  {
    pthread_t *_t = _make_pthread_t("tokenGen");
    struct _tokenGen_args *_margs = malloc(sizeof(struct _tokenGen_args));
    struct _tokenGen_args _args = {chan2, int2};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, tokenGen, (void *)_margs);
  };
  {
    pthread_t *_t = _make_pthread_t("interleaver");
    struct _interleaver_args *_margs = malloc(sizeof(struct _interleaver_args));
    struct _interleaver_args _args = {chan1, chan2, chan3};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, interleaver, (void *)_margs);
  };
  {
    pthread_t *_t = _make_pthread_t("printer");
    struct _printer_args *_margs = malloc(sizeof(struct _printer_args));
    struct _printer_args _args = {chan3};
    memcpy((void *)_margs, (void *)&_args, sizeof(typeof(_args)));
    pthread_create(_t, NULL, printer, (void *)_margs);
  };
  _wait_for_finish(false);
}
```

7 Lessons Learned

7.1 Adam

When in doubt, restart your computer. When dealing with software that creates a lot of threads, you can run into strange resource limit issues that are easily resolved by starting with a clean system. Draw and hand simulate algorithms you don't understand. Bitonic sort seemed very alien to me until I started drawing it out. When programming in a different paradigm (object-oriented, functional, dataflow), it seems hard at first, but you just have to think about problems in an entirely different way. Bitonic sort and the fibonacci program demonstrate this.

7.2 Zach

Overly broad github issues such as “clean code” will never be closed. Writing tests that break the program is a much better way to prioritize things that need to be fixed. If your team has a designated time to meet, make sure the group meets. Saying “let’s just work individually” can be interpreted as “I have other work I need to do.” If one or two group members cannot make a meeting, still meet, and do not cancel. Doing a little each week keeps momentum going and is much more effective than pushing back work for another week. Make sure everyone has something to work on. No one should be waiting on another group member.

7.3 Mitchell

Much of our time seemed at first to be unproductive: we spent 80% of it talking, planning, and brainstorming. 20% of our time together was spent on programming. But, I came to realize that communication is VERY important. Everyone needs to be on the same page. Discussion often exposes potential pitfalls, thereby lessening the group’s chances of succumbing to them.

In addition, I realized that writing a compiler is an emergent phenomenon. One second, you have a bunch of crazy disconnected parts that don’t run, and in the next, they somehow work together to become a compiler. There’s no reason to get discouraged if the task seems daunting; a tiny step every day will bring all of the moving parts together eventually. The most important thing is the test suite. It holds everything together like glue.

7.4 Hyonjee

Setting up two regular weekly meeting times at the beginning of the semester really helped our group make consistent progress throughout the duration of the project. It was also very helpful to discuss design and come up with a general implementation plan before writing the corresponding code. This ensured that our group was on the same page and allowed us to develop parts of the system simultaneously. Lastly, we would not have been successful without a solid test framework. With multiple people contributing code, a thorough test suite was the most effective way to make sure we didn’t break things in the system. Tests also gave us tangible goals and direction as we neared the end of our project.

8 Appendix

Table of Contents & Authorship

8.1 scanner.mll
- Primary contributor(s): Zach and Adam
- Secondary contributor(s): Mitchell and Hyonjee

8.2 ast.ml
- Primary contributor(s): Mitchell

8.3 parser.mly
- Primary contributor(s): Adam and Mitchell
- Secondary Contributor(s): Zach and Hyonjee

8.4 sast.ml
- Primary contributor(s): Mitchell

8.5 semantic_analysis.ml
- Primary contributor(s): Mitchell

- Secondary contributor(s): Zach, and Hyonjee

8.6 compile.ml

- Primary Contributor(s): Mitchell
- Secondary contributor(s): Hyonjee and Zach

8.7 c_runtime.c

- Primary Contributors: Mitchell and Hyonjee

8.8 flowc.ml

- Primary Contributor(s): Adam
- Secondary Contributor(s): Mitchell

Omitted from Appendix

bitonic sort

- Primary Contributor(s): Adam

tests

- Primary Contributor(s): Zach
- Secondary Contributors(s): Mitchell, Hyonjee, Adam

8.1 scanner.mll

```
{ open Parser }

(* Definitions *)

let digit = ['0'-'9']
let double = ((digit+ '.' digit*) | ('.' digit+))

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "//"           { comment lexbuf }           (* Comments *)
| '.'            { DOT }
| '('            { LPAREN }
| ')'            { RPAREN }
| '{'            { LBRACE }
| '}'            { RBRACE }
| '['            { LBRACKET}
| ']'            { RBRACKET}
| ';'            { SEMI }
| ','            { COMMA }
| '+'            { PLUS }
| '-'            { MINUS }
| '*'            { TIMES }
| '/'            { DIVIDE }
| '='            { ASSIGN }
| '#'            { LIST_LENGTH }
| "^"            { LIST_TAIL }
| "=="           { EQ }
| "!="           { NEQ }
| '<'            { LT }
| "<="           { LEQ }
| ">"            { GT }
| ">="           { GEQ }
| '!'            { NOT }
| "||"           { OR }
| "&&"           { AND }
| '%'            { MODULO }
| '@'            { RETRIEVE }
| "->"           { WRITE_CHANNEL }
| "::"           { CONCAT }
| "if"           { IF }
| "else"         { ELSE }
| "for"          { FOR }
| "while"        { WHILE }
```

```
| "return"      { RETURN }
| "poison"        { POISON }
| "int"         { INT }
| "double"      { DOUBLE }
| "char"        { CHAR }
| "bool"        { BOOL }
| "break"       { BREAK }
| "continue"    { CONTINUE }
| "string"      { STRING }
| "list"        { LIST }
| "in"          { IN }
| "out"         { OUT }
| "channel"     { CHANNEL }
| "proc"        { PROC }
| "void"        { VOID }
| "true"        { BOOL_LITERAL(true) }
| "false"       { BOOL_LITERAL(false) }
| digit+ as lxm { INT_LITERAL(int_of_string lxm) }
| double as lxm { DOUBLE_LITERAL(float_of_string lxm)}
| '\"' ([^'\"']* as lxm) '\"' { STRING_LITERAL(lxm) }
| '\'' ([' '-'&' '('-'[' ']'-'~'] as lxm) '\'' { CHAR_LITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { IDENTIFIER(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  '\n' { token lexbuf }
| _    { comment lexbuf }
```

8.2 ast.ml

```
(* ast.ml *)
type bin_op =
  | Plus
  | Minus
  | Times
  | Divide
  | Modulo
  | Neq
  | Lt
  | Leq
  | Gt
  | Geq
  | Eq
  | Send
  | And
  | Or
  | Assign
  | Concat

type unary_op = | Retrieve | Negate | Not | ListLength | ListTail

type direction = | In | Out | Nodir

(* All of the primitive and nonprimitive types *)
type flow_type =
  | Int
  | Double
  | Bool
  | Char
  | Void
  | Proc
  | String
  | Channel of flow_type * direction
  | List of flow_type

```

```
type dot_initializer =
  { dot_initializer_id : string; dot_initializer_val : expr
  }
  and expr =
  | IntLiteral of int
  | StringLiteral of string
  | BoolLiteral of bool
  | CharLiteral of char
  | DoubleLiteral of float
  | ListInitializer of expr list
  | Id of string
  | BinOp of expr * bin_op * expr
  | UnaryOp of unary_op * expr
  | FunctionCall of string * expr list
  | Noexpr

type variable_declaration =
  { declaration_type : flow_type; declaration_id : string;
    declaration_initializer : expr
  }

type stmt =
  | Expr of expr
  | Block of stmt list
  | Return of expr
  | Declaration of variable_declaration
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Continue
  | Break
  | Poison of expr

type function_declaration =
  { return_type : flow_type; function_name : string;
    arguments : variable_declaration list; has_definition : bool;
    body : stmt list
  }

type declaration =
  | VarDecl of variable_declaration | FuncDecl of function_declaration

type program = declaration list
```

8.3 parser.mly

```
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN CONCAT
%token WRITE_CHANNEL RETRIEVE PROC CHANNEL IN OUT
%token BREAK CONTINUE VOID
%token POISON
%token OR AND NOT
%token DOUBLE CHAR BOOL INT STRING LIST
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR WHILE
%token LIST_LENGTH LIST_TAIL
%token <int> INT_LITERAL
%token <float> DOUBLE_LITERAL
%token <char> CHAR_LITERAL
%token <bool> BOOL_LITERAL
%token <string> STRING_LITERAL
%token <string> IDENTIFIER
%token EOF

```

```
%nonassoc NOELSE /* dummy variable for lowest precedence */
%nonassoc ELSE
%left WRITE_CHANNEL
%right ASSIGN
%left AND OR
%left EQ NEQ
%left LT GT LEQ GEQ
%right CONCAT
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%left RETRIEVE
%nonassoc UNARY_OP  /* dummy variable for highest precedence */

%start program
%type <Ast.program> program

%%

program:
  declarations EOF {List.rev $1}

declarations:
    /* nothing */ { [] }
  | declarations var_declaration SEMI { VarDecl($2)::$1  }
  | declarations function_declaration { FuncDecl($2)::$1 }

function_declaration:
    flow_type IDENTIFIER LPAREN arg_decl_list RPAREN LBRACE stmt_list RBRACE
    {
        {
            return_type = $1;
            function_name = $2;
            arguments = $4;
            has_definition = true;
            body = $7;
        }
    }
  | flow_type IDENTIFIER LPAREN arg_decl_list RPAREN LBRACE RBRACE
    {
      {
            return_type = $1;
            function_name = $2;
            arguments = $4;
            has_definition = false;
            body = [];
      }
    }

arg_decl_list:
    /* nothing */ {[]}
  | arg_decl {[$1]}
  | arg_decl COMMA arg_decl_list {$1::$3}

arg_decl:
    simple_var_declaration {$1}

flow_type:
    INT                     {Int}
  | DOUBLE                  {Double}
  | CHAR                    {Char}
  | BOOL                    {Bool}
  | VOID                    {Void}
  | PROC                    {Proc}
  | STRING                  {String}
  | LIST LT flow_type GT    {List($3)}
  | CHANNEL LT flow_type GT {Channel($3, Nodir)}
  | IN flow_type            {Channel($2, In)}
  | OUT flow_type           {Channel($2, Out)}

var_declaration:
    simple_var_declaration      {$1}
  | init_var_declaration        {$1}
```

```
 93
 94 simple_var_declaration:
 95     flow_type IDENTIFIER {{declaration_type = $1;
 96                            declaration_id   = $2;
 97                            declaration_initializer = Noexpr}}
 98
 99 init_var_declaration:
100   | flow_type IDENTIFIER ASSIGN expr {{declaration_type = $1;
101                                       declaration_id   = $2;
102                                       declaration_initializer = $4}}
103
104 stmt_list:
105     stmt {[$1]}
106   | stmt stmt_list {$1::$2}
107
108 stmt:
109     expr_stmt        {$1}
110   | compound_stmt    {$1}
111   | selection_stmt   {$1}
112   | iteration_stmt   {$1}
113   | var_declaration SEMI {Declaration($1)}
114   | jump_stmt        {$1}
115   | poison_stmt      {$1}
116
117 expr_stmt:
118     expr SEMI {Expr($1)}
119
120 compound_stmt:
121     LBRACE stmt_list RBRACE {Block($2)}
122   | LBRACE RBRACE {Block([])}
123
124 selection_stmt:
125     IF LPAREN expr RPAREN stmt %prec NOELSE {If($3, $5, Expr(Noexpr))}
126   | IF LPAREN expr RPAREN stmt ELSE stmt    {If($3, $5, $7)}
127
128 iteration_stmt:
129     WHILE LPAREN expr RPAREN stmt {While($3, $5)}
130   | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt {For($3, $5, $7, $9)}
131
132 jump_stmt:
133     RETURN expr SEMI {Return($2)}
134   | RETURN SEMI {Return(Noexpr)}
135   | CONTINUE SEMI {Continue}
136   | BREAK SEMI {Break}
137
138 poison_stmt:
139     POISON IDENTIFIER SEMI {Poison(Id($2))}
140
141 expr_opt:
142     /* nothing */ {Noexpr}
143   | expr {$1}
144
145 expr_list:
146     expr {[$1]}
147   | expr COMMA expr_list {$1::$3}
148
149 expr:
150     INT_LITERAL {IntLiteral($1)}
151   | DOUBLE_LITERAL {DoubleLiteral($1)}
152   | STRING_LITERAL {StringLiteral($1)}
153   | CHAR_LITERAL {CharLiteral($1)}
154   | BOOL_LITERAL {BoolLiteral($1)}
155   | IDENTIFIER {Id($1)}
156   | LBRACKET expr_list RBRACKET {ListInitializer($2)}
157   | LBRACKET RBRACKET {ListInitializer([])}
158   | RETRIEVE IDENTIFIER {UnaryOp(Retrieve, Id($2))}
159   | expr WRITE_CHANNEL IDENTIFIER {BinOp($1, Send, Id($3))}
160   | function_call {$1}
161   | expr CONCAT expr {BinOp($1, Concat, $3)}
162   | expr PLUS expr {BinOp($1, Plus, $3)}
163   | expr MINUS expr {BinOp($1, Minus, $3)}
164   | expr TIMES expr {BinOp($1, Times, $3)}
```

```
165   | expr DIVIDE expr {BinOp($1, Divide, $3)}
166   | expr MODULO expr {BinOp($1, Modulo, $3)}
167   | expr EQ expr {BinOp($1, Eq, $3)}
168   | expr NEQ expr {BinOp($1, Neq, $3)}
169   | expr LT expr {BinOp($1, Lt, $3)}
170   | expr GT expr {BinOp($1, Gt,$3)}
171   | expr LEQ expr {BinOp($1, Leq,$3)}
172   | expr GEQ expr {BinOp($1, Geq,$3)}
173   | expr AND expr {BinOp($1, And,$3)}
174   | expr OR expr {BinOp($1, Or,$3)}
175   | IDENTIFIER ASSIGN expr {BinOp(Id($1), Assign, $3)}
176   | LPAREN expr RPAREN {$2}
177   | NOT expr %prec UNARY_OP {UnaryOp(Not, $2)}
178   | MINUS expr %prec UNARY_OP { UnaryOp(Negate, $2) }
179   | LIST_LENGTH expr %prec UNARY_OP { UnaryOp(ListLength, $2)}
180   | LIST_TAIL expr %prec UNARY_OP { UnaryOp(ListTail, $2)}
181
182 function_call:
183     IDENTIFIER LPAREN RPAREN {FunctionCall($1, [])}
184   | IDENTIFIER LPAREN expr_list RPAREN {FunctionCall($1, $3)}
```

8.4 sast.ml

```
 1 (* sast.ml *)
 2 open Ast
 3
 4 type s_dot_initializer =
 5   { s_dot_initializer_id : string; s_dot_initializer_val : typed_expr
 6   }
 7
 8 (* typed expression *)
 9 and expr_details =
10   | TIntLiteral of int
11   | TStringLiteral of string
12   | TBoolLiteral of bool
13   | TCharLiteral of char
14   | TDoubleLiteral of float
15   | TListInitializer of typed_expr list
16   | TId of string
17   | TBinOp of typed_expr * bin_op * typed_expr
18   | TUnaryOp of unary_op * typed_expr
19   | TFunctionCall of string * typed_expr list
20   | TNoexpr
21
22 and typed_expr = (expr_details * flow_type)
23
24 type s_variable_declaration =
25   { s_declaration_type : flow_type; s_declaration_id : string;
26     s_declaration_initializer : typed_expr
27   }
28
29 type s_stmt =
30   | SExpr of typed_expr
31   | SBlock of s_stmt list
32   | SReturn of typed_expr
33   | SDeclaration of s_variable_declaration
34   | SIf of typed_expr * s_stmt * s_stmt
35   | SFor of typed_expr * typed_expr * typed_expr * s_stmt
36   | SWhile of typed_expr * s_stmt
37   | SContinue
38   | SBreak
39   | SPoison of typed_expr
40   | SExitProc
41
```

```
type s_function_declaration =
  { s_return_type : flow_type; s_function_name : string;
    s_arguments : s_variable_declaration list; s_has_definition : bool;
    s_body : s_stmt list
  }

type s_declaration =
  | SVarDecl of s_variable_declaration | SFuncDecl of s_function_declaration

type s_program = s_declaration list


```

8.5 semantic_analysis.ml

```
(* semantic_analysis.ml - take an ast and produces an sast *)
open Ast
open Sast

type symtab =
  { parent : symtab option; variables : variable_declaration list
  }

type function_entry =
  { name : string; param_types : flow_type list; ret_type : flow_type
  }

type environment =
  { return_type : flow_type option; symbol_table : symtab;
    funcs : function_entry list; in_loop : bool
  }

let check_progam (prog : program) : s_program =
  let rec find_variable_decl (symbol_table : symtab) (name : string) :
    variable_declaration =
    try
      List.find (fun var_decl -> var_decl.declaration_id = name)
        symbol_table.variables
    with
    | Not_found ->
        (match symbol_table.parent with
         | Some parent -> find_variable_decl parent name
         | _ -> raise Not_found) in
  let find_variable_type (symbol_table : symtab) (name : string) :
    flow_type =
    let vdecl = find_variable_decl symbol_table name
    in vdecl.declaration_type in
  let is_logical (expr : typed_expr) : bool =
    match expr with
    | (_, Int) | (_, Bool) | (_, Char) | (_, String) | (_, Double) -> true
    | (_, Channel (t, dir)) -> true
    | _ -> false in
  let is_arithmetic (expr : typed_expr) : bool =
    match expr with | (_, Int) | (_, Double) -> true | _ -> false in
  let string_of_binop =
    function
    | Plus -> "+"
    | Minus -> "-"
    | Times -> "*"
    | Divide -> "/"
    | Modulo -> "%"
    | Neq -> "!="
    | Lt -> "<"
    | Leq -> "<="
    | Gt -> ">"
    | Geq -> ">="
```

```
52      | Eq -> "=="
53      | Send -> "->"
54      | And -> "&&"
55      | Or -> "||"
56      | Concat -> "::"
57      | Assign -> "=" in
58    let string_of_unop =
59      function
60      | Retrieve -> "@"
61      | Negate -> "-"
62      | Not -> "!"
63      | ListLength -> "#"
64      | ListTail -> "^" in
65    let rec string_of_type =
66      function
67      | Int -> "int"
68      | Double -> "double"
69      | Bool -> "bool"
70      | Char -> "char"
71      | Void -> "void"
72      | Proc -> "proc"
73      | String -> "string"
74      | Channel (t, Nodir) -> "channel<" ^ ((string_of_type t) ^ ">")
75      | Channel (t, In) -> "in " ^ (string_of_type t)
76      | Channel (t, Out) -> "out " ^ (string_of_type t)
77      | List t -> "list<" ^ ((string_of_type t) ^ ">") in
78    (* Helper function to check if a variable is a previously declared global *)
79    let rec is_declared_global_var (name: string) (symbol_table: symtab) : bool =
80      if (List.exists (fun var_decl -> var_decl.declaration_id = name) symbol_table.variables)
81      then (match symbol_table.parent with
82            | Some parent -> false
83            | _ -> true)
84      else (match symbol_table.parent with
85            | Some parent -> is_declared_global_var name parent
86            | _ -> false)
87    in
88    let check_binop (e1 : typed_expr) (e2 : typed_expr) (op : bin_op) (env: environment) :
89      typed_expr =
90      let (expr_details1, t1) = e1
91      and (expr_details2, t2) = e2
92      in
93        match op with
94        | Plus | Minus | Times | Divide | Modulo | Lt | Leq | Gt | Geq ->
95            if (is_arithmetic e1) && (is_arithmetic e2)
96            then
97              (let final_type =
98                 if (t1 = Double) || (t2 = Double) then Double else Int
99               in ((TBinOp (e1, op, e2)), final_type))
100           else
101             raise
102               (Invalid_argument
103                  ("operator " ^
104                     ((string_of_binop op) ^
105                        (" not compatible with " ^
106                           ((string_of_type t1) ^
107                              (" and " ^ (string_of_type t2)))))))
108       | And | Or | Eq | Neq ->
109           if (is_logical e1) && (is_logical e2)
110           then ((TBinOp (e1, op, e2)), Bool)
111           else
112             raise
113               (Invalid_argument
114                  ("Attempting a logical operation" ^ "on invalid operands"))
115       | Assign ->
116           (match expr_details1 with
117            | TId name ->
118                if t1 = t2
119                then (if (is_declared_global_var name env.symbol_table)
120                      then (raise (Failure "Global variables are immutable"))
121                     else((TBinOp (e1, op, e2)), t1))
122                else
123                  raise
```

```
124                   (Invalid_argument
125                      "Identifier type does not match expression")
126           | _ -> raise (Invalid_argument "Attempting assignment to non-id"))
127       | Send ->
128           (match t2 with
129            | Channel (t, Out) when t = t1 -> ((TBinOp (e1, op, e2)), t1)
130            | _ -> raise (Invalid_argument "Invalid write to channel"))
131       | Concat ->
132           (match (t1, t2) with
133            | (_, List t) ->
134                if t = t1
135                then ((TBinOp (e1, op, e2)), t2)
136                else raise (Failure "Type mismatch for list operation.")
137            | (_, _) -> raise (Failure "Can only concat to front of list.")) in
138   let check_unop (e : typed_expr) (op : unary_op) : typed_expr =
139     let (_, t) = e
140     in
141       match op with
142       | Retrieve ->
143           (match t with
144            | (* Only In channels can be operated on by @ operator. *)
145                Channel (t, In) -> ((TUnaryOp (op, e)), t)
146            | List list_type -> ((TUnaryOp (op, e)), list_type)
147            | _ ->
148                raise
149                  (Invalid_argument
150                     ("operator " ^
151                        ((string_of_unop op) ^
152                           (" not compatible with " ^ (string_of_type t))))))
153       | Negate ->
154           (match t with
155            | Int | Double -> ((TUnaryOp (op, e)), t)
156            | _ ->
157                raise
158                  (Invalid_argument
159                     ("operator " ^
160                        ((string_of_unop op) ^
161                           (" not compatible with " ^ (string_of_type t))))))
162       | ListLength ->
163           (match t with
164            | List _ -> ((TUnaryOp (op, e)), Int)
165            | _ ->
166                raise
167                  (Invalid_argument
168                     ("operator " ^
169                        ((string_of_unop op) ^
170                           (" not compatible with " ^ (string_of_type t))))))
171       | ListTail ->
172           (match t with
173            | List _ -> ((TUnaryOp (op, e)), t)
174            | _ ->
175                raise
176                  (Invalid_argument
177                     ("operator " ^
178                        ((string_of_unop op) ^
179                           (" not compatible with " ^ (string_of_type t))))))
180       | Not ->
181           (* Channels and such can be operated on by the negation operator *)
182           if is_logical e
183           then ((TUnaryOp (op, e)), Bool)
184           else
185             raise
186               (Invalid_argument
187                  ("operator " ^
188                     ((string_of_unop op) ^
189                        (" not compatible with " ^ (string_of_type t))))) in
190   let string_of_type_list type_list =
191     List.fold_left (fun acc elm -> acc ^ (", " ^ (string_of_type elm)))
192       (string_of_type (List.hd type_list)) (List.tl type_list) in
193   let string_of_actual_list actual_list =
194     List.fold_left (fun acc elm -> acc ^ (", " ^ (string_of_type (snd elm))))
195       (string_of_type (snd (List.hd actual_list))) (List.tl actual_list) in
```

```
196   (* Should get consolidated *)
197   let built_in_funcs =
198     [ { name = "print_string"; param_types = [ String ]; ret_type = Void; };
199       { name = "print_int"; param_types = [ Int ]; ret_type = Void; };
200       { name = "print_char"; param_types = [ Char ]; ret_type = Void; };
201       { name = "print_double"; param_types = [ Double ]; ret_type = Void; };
202       { name = "println"; param_types = []; ret_type = Void; };
203       { name = "rand"; param_types = []; ret_type = Double; } ] in
204   let check_function_call (name : string) (actual_list : typed_expr list)
205     (env : environment) : typed_expr =
206     try
207       (* Attempt to find the function in the current environment *)
208       let f_entry = List.find (fun f -> f.name = name) env.funcs in
209       (* Get rid of channel directions, for the purpose of
210          * parameter matching *)
211       let no_dir_param_types =
212         List.map
213           (fun p_type ->
214              match p_type with
215              | Channel (ft, dir) -> Channel (ft, Nodir)
216              | _ -> p_type)
217           f_entry.param_types in
218       (* If not a built in function, it should be one to one match.*)
219       let actual_param_types =
220         List.map (fun texp -> let (e, t) = texp in t) actual_list
221       in
222         if
223           (no_dir_param_types <> actual_param_types) &&
224             (f_entry.param_types <> actual_param_types)
225         then
226           raise
227             (Failure
228                ("Incorrect paramater types for function call " ^
229                   (name ^
230                      (". param types: " ^
231                         ((string_of_type_list f_entry.param_types) ^
232                            (". actual types: " ^
233                               (string_of_actual_list actual_list)))))))
234         else ((TFunctionCall (name, actual_list)), (f_entry.ret_type))
235     with | Not_found -> raise (Failure ("Undeclared function " ^ name)) in
236   (* Expressions never return a new environment since they can't mutate the
237  * environments *)
238   let rec check_expr (env : environment) (e : expr) : typed_expr =
239     match e with
240     | IntLiteral i -> ((TIntLiteral i), Int)
241     | StringLiteral s -> ((TStringLiteral s), String)
242     | BoolLiteral b -> ((TBoolLiteral b), Bool)
243     | CharLiteral c -> ((TCharLiteral c), Char)
244     | DoubleLiteral d -> ((TDoubleLiteral d), Double)
245     | Id s -> (* Try to find the variable in the symbol table *)
246         let t =
247           (try find_variable_type env.symbol_table s
248            with | Not_found -> raise (Failure ("Undeclared identifier " ^ s)))
249         in ((TId s), t)
250     | BinOp (e1, op, e2) ->
251         let checked_e1 = check_expr env e1
252         and checked_e2 = check_expr env e2
253         in check_binop checked_e1 checked_e2 op env
254     | ListInitializer expr_list ->
255         if (List.length expr_list) == 0
256         then (TNoexpr, Void)
257         else
258           (let checked_expr_list =
259              List.map (fun exp -> check_expr env exp) expr_list in
260            let list_type = snd (List.hd checked_expr_list) in
261            let of_same_type =
262              List.for_all
263                (fun e -> if (snd e) = list_type then true else false)
264                checked_expr_list
265            in
266              if of_same_type
267              then ((TListInitializer checked_expr_list), (List list_type))
```

```
268             else
269               raise
270                 (Failure
271                    "List must be initialized with expressions of the same type"))
272     | UnaryOp (unary_op, e) ->
273         let checked_expr = check_expr env e
274         in check_unop checked_expr unary_op
275     | FunctionCall (name, actual_list) ->
276         check_function_call name
277           (List.map (fun exp -> check_expr env exp) actual_list) env
278     | Noexpr -> (TNoexpr, Void) in
279   let check_variable_declaration (env : environment)
280                                  (decl : variable_declaration) =
281     let (expr_details, t) = check_expr env decl.declaration_initializer
282     in
283       (* Either the expression needs to match the declaration's type, or it
284        * can be Noexpr (which is void) *)
285       if (t = decl.declaration_type) || (t = Void)
286       then
287         (try
288            let _ =
289              (* Try to find the a local variable of the same name. If found, it's an error. *)
290              List.find
291                (fun vdecl -> vdecl.declaration_id = decl.declaration_id)
292                env.symbol_table.variables
293            in
294              raise
295                (Failure
296                   ("Variable " ^
297                      (decl.declaration_id ^
298                         " already declared in local scope")))
299          with
300          | (* If not found, add the declaration to the symbol table and return the new environment *)
301              Not_found ->
302              let new_symbol_table =
303                {
304                  (env.symbol_table)
305                  with
306                  variables = decl :: env.symbol_table.variables;
307                } in
308              let new_env = { (env) with symbol_table = new_symbol_table; }
309              and s_var_decl =
310                {
311                  s_declaration_type = decl.declaration_type;
312                  s_declaration_id = decl.declaration_id;
313                  s_declaration_initializer = (expr_details, t);
314                }
315              in (new_env, s_var_decl))
316       else
317         raise
318           (Failure
319              (decl.declaration_id ^
320                 (": Declaration type does not match expression\n" ^
321                    ("Attempting to initialize " ^
322                       ((string_of_type decl.declaration_type) ^
323                          (" with " ^ (string_of_type t)))))))) in
324   let check_arg_declaration (env : environment) (decl : variable_declaration)
325                             =
326     match decl.declaration_initializer with
327     | Noexpr ->
328         check_variable_declaration env decl
329     | _ ->
330         raise
331           (Failure
332              ("Error in argument declaration for " ^
333                 (decl.declaration_id ^
334                    ": Cannot have default values in function declaration."))) in
335   let rec check_stmt (env : environment) (stmt : stmt) :
336     (environment * s_stmt) =
337     match stmt with
338     | (* Expressions cannot mutate the environment, so the current env is
339          * returned *)
```

```
340         Expr e -> (env, (SExpr (check_expr env e)))
341     | (* Blocks have their own scope, so the environment must be the same
342          * after the block has been semantically analyzed. Hence, as with
343          * Expr, we return the current env. *)
344         Block stmt_list ->
345         let new_symbol_table =
346           { parent = Some env.symbol_table; variables = []; } in
347         let (_, checked_stmts) =
348           check_stmt_list { (env) with symbol_table = new_symbol_table; }
349             stmt_list false
350         in (env, (SBlock checked_stmts))
351     | (* A return statement must have the same return type as the
352          * one we're expecting. Recall that the return type is set before
353          * entering a function. *)
354         Return e ->
355         let (expr_details, t) = check_expr env e
356         in
357           (match env.return_type with
358            | Some Proc ->
359                if t = Void
360                then (env, SExitProc)
361                else raise (Failure "Atempting return value from process")
362            | Some rtype ->
363                if t = rtype
364                then (env, (SReturn (check_expr env e)))
365                else raise (Failure "Expression does not match return_type")
366            | None -> raise (Failure "Return statement not in function"))
367     | (* Declarations WILL mutate the environment, so we
368          * return the new environment. *)
369         Declaration vdecl ->
370         let (new_env, vdecl) = check_variable_declaration env vdecl
371         in (new_env, (SDeclaration vdecl))
372     | (* The restriction on the expression in an if statement is that
373          * it must be logical (truey or falsey) *)
374         If (e, s1, s2) ->
375         let checked_expr = check_expr env e
376         and (_, checked_stmt1) = check_stmt env s1
377         and (_, checked_stmt2) = check_stmt env s2
378         in
379           if is_logical checked_expr
380           then (env, (SIf (checked_expr, checked_stmt1, checked_stmt2)))
381           else raise (Failure "Invalid expression in \"if\" statement")
382     | (* Similar restrictions as for if statments. However, we must additionally
383          * make sure to set the environment's in_loop variable before checking
384          * the statements (in case the statements include a break or continue *)
385         For (e1, e2, e3, s) ->
386         let checked_expr1 = check_expr env e1
387         and checked_expr2 = check_expr env e2
388         and checked_expr3 = check_expr env e3
389         and (_, checked_stmt) = check_stmt { (env) with in_loop = true; } s
390         in
391           if
392             (is_logical checked_expr1) &&
393               ((is_logical checked_expr2) && (is_logical checked_expr3))
394           then
395             (env,
396              (SFor (checked_expr1, checked_expr2, checked_expr3,
397                 checked_stmt)))
398           else raise (Failure "Invalid expression in \"for\" statement")
399     | While (e, s) ->
400         let checked_expr = check_expr env e
401         and (_, checked_stmt) = check_stmt { (env) with in_loop = true; } s
402         in
403           if is_logical checked_expr
404           then (env, (SWhile (checked_expr, checked_stmt)))
405           else raise (Failure "Invalid expression in \"while\" statement")
406     | (* Continue and break statements don't make sense outside of a loop *)
407         Continue ->
408         if env.in_loop = true
409         then (env, SContinue)
410         else raise (Failure "Not in a loop")
411     | Break ->
```

```
412         if env.in_loop = true
413         then (env, SBreak)
414         else raise (Failure "Not in a loop")
415     | (* Only "out" channels can be poisoned from inside a process. *)
416         Poison e ->
417         let (expr_details, t) = check_expr env e
418         in
419           (match t with
420            | Channel (t, Out) ->
421                (env, (SPoison (expr_details, (Channel (t, Out)))))
422            | Channel (t, _) -> raise (Failure "Can only poison out channels")
423            | _ -> raise (Failure "Attempting to poison a non-channel"))
424   and check_stmt_list (env : environment) (stmt_list : stmt list)
425     (must_return : bool) : (environment * (s_stmt list)) =
426     (* The environments have to be folded through the stmt list.
427      * Each statement takes the updated environment generated from
428      * the last one. acc (the accumulator) is a pair of env, checked
429      * statements. The statements must be reversed because they are collected
430      * backwards in a list. *)
431     let _ =
432       if must_return
433       then
434         (try
435            ignore
436              (List.find
437                 (fun s -> match s with | Return _ -> true | _ -> false)
438                 stmt_list)
439          with
440          | Not_found -> raise (Failure "Non-void function might not return"))
441       else () in
442     let (new_env, checked_stmts) =
443       List.fold_left
444         (fun acc stmt ->
445            let (env', stmt_node) = check_stmt (fst acc) stmt
446            in (env', (stmt_node :: (snd acc))))
447         (env, []) stmt_list
448     in (new_env, (List.rev checked_stmts)) in
449   let check_function_declaration (env : environment)
450                                  (fdecl : function_declaration) =
451     (* Get the types of the function's parameters *)
452     let p_types =
453       List.map (fun vdecl -> vdecl.declaration_type) fdecl.arguments in
454     (* Make a function entry for the current function *)
455     let f_entry =
456       {
457         name = fdecl.function_name;
458         param_types = p_types;
459         ret_type = fdecl.return_type;
460       } in
461     let new_funcs = f_entry :: env.funcs in
462     (* Make a new symbol table for the function scope *)
463     let new_symbol_table =
464       { parent = Some env.symbol_table; variables = []; } in
465     (* Add the function currently being checked to the environment. This is
466      * needed in the case of recursion (ie encountering a function call
467      * referencing this function in the body). Furthermore, set the return
468      * type and symbol table with now-empty local scope *)
469     let new_env =
470       {
471         (env)
472         with
473         funcs = new_funcs;
474         return_type = Some fdecl.return_type;
475         symbol_table = new_symbol_table;
476       } in
477     (* Get the arguments into the scope by folding the environment
478      * over the parameter list *)
479     let (env_with_args, arg_decl_list) =
480       List.fold_left
481         (fun acc arg_decl ->
482            let (env', arg_node) = check_arg_declaration (fst acc) arg_decl
483            in (env', (arg_node :: (snd acc))))
```

```
        (new_env, []) fdecl.arguments in
    (* Check the function body. Discard the environment. We won't need it
     * outside the scope of the function body. *)
    let must_return =
      if
        (fdecl.return_type = Void) ||
          ((fdecl.return_type = Proc) || (fdecl.function_name = "main"))
      then false
      else true in
    let (_, func_body) =
      check_stmt_list env_with_args fdecl.body must_return in
    let func_body =
      if ( != ) fdecl.return_type Proc
      then func_body
      else func_body @ [ SExitProc ] in
    (* Create the function node to return *)
    let func_node =
      {
        s_return_type = fdecl.return_type;
        s_function_name = fdecl.function_name;
        s_arguments = List.rev arg_decl_list;(* We collected list in reverse order *)
        s_has_definition = true;
        s_body = func_body;
      }
    in
      (* Return the original environment, with the current function appended *)
      ({ (env) with funcs = new_funcs; }, func_node) in
  (* Check declaration returns a new environment, which is populated with the
 * newly declared symbol *)
  let check_declaration (env : environment) (decl : declaration) :
    (environment * s_declaration) =
    match decl with
    | VarDecl vdecl ->
        let (new_env, checked_vdecl) = check_variable_declaration env vdecl
        in (new_env, (SVarDecl checked_vdecl))
    | FuncDecl fdecl ->
        let (new_env, checked_fdecl) = check_function_declaration env fdecl
        in (new_env, (SFuncDecl checked_fdecl)) in
  (* Here, we set up the initial environment. The return type is None, meaning
 * that we have yet to descend into semantically analyzing functions. The
 * symbol table is at the top level and has no parent. The functions in the
 * current scope are only the built in ones. *)
  let env =
    {
      return_type = None;
      symbol_table = { parent = None; variables = []; };
      funcs = built_in_funcs;
      in_loop = false;
    } in
  (* acc is the accumulator it's a tuple of env, decl_list.
 * the fold left builds the accumulator, threading the environment
 * through the list of declarations. When the fold finishes, decl_list
 * should be a built list of s_declarations *)
  let (_, decl_list) =
    List.fold_left
      (fun acc decl ->
         let (new_env, snode) = check_declaration (fst acc) decl
         in (new_env, (snode :: (snd acc))))
      (env, []) prog
  in decl_list
```

8.6 compile.ml

```
1 (* Compile.ml - takes a full sast and compiles it to c code *)
```

```
 2 open Ast
 3 open Sast
 4 open Boilerplate
 5
 6 let supported_channels = [ Int; Char; Double ]
 7 let supported_lists = [ Int; Char; Double; Channel (Int, Nodir) ]
 8
 9 let compile (program : s_program) (dot : bool) : string =
10
11   (* Toggle whether the resulting c program should print a dot graph *)
12   let print_dot = if dot then "true" else "false" in
13
14   (* Translate flow type to c type *)
15   let rec translate_type (ftype : flow_type) =
16     match ftype with
17     | Int -> "int"
18     | Double -> "double"
19     | Bool -> "int"
20     | Char -> "char"
21     | Void -> "void"
22     | Proc -> "void*"
23     | String -> "char *"
24     | Channel (t, dir) ->
25         (try
26            let _ = List.find (fun e -> t = e) supported_channels
27            in "struct _" ^ ((translate_type t) ^ "_channel* ")
28          with | Not_found -> raise (Failure "Channel not supported"))
29     | List t ->
30         (try
31            let _ = List.find (fun e -> t = e) supported_lists
32            in "struct _cell *"
33          with | Not_found -> raise (Failure "List not supported")) in
34
35   (* Wrap channels used in a logical context in _wait_for_more *)
36   let wait_for_more (exp: string) (t: flow_type) : string =
37     match t with
38     | Channel (_, _) -> "_wait_for_more( (struct _channel*) " ^ (exp ^ ")")
39     | _ -> exp in
40
41   (* Translate a flow expression *)
42   let rec translate_expr (expr : typed_expr) : string =
43     let translate_bin_op (typed_exp1 : typed_expr) (bin_op : bin_op)
44                          (typed_exp2 : typed_expr) =
45       let t1 = snd typed_exp1
46       and t2 = snd typed_exp2
47       and exp1 = translate_expr typed_exp1
48       and exp2 = translate_expr typed_exp2
49       in
50         match bin_op with
51         | Plus -> exp1 ^ ("+" ^ exp2)
52         | Minus -> exp1 ^ ("-" ^ exp2)
53         | Times -> exp1 ^ ("*" ^ exp2)
54         | Divide -> exp1 ^ ("/" ^ exp2)
55         | Modulo -> exp1 ^ ("%" ^ exp2)
56         | Eq -> exp1 ^ ("==" ^ exp2)
57         | Neq -> exp1 ^ ("!=" ^ exp2)
58         | Lt -> exp1 ^ ("<" ^ exp2)
59         | Gt -> exp1 ^ (">" ^ exp2)
60         | Leq -> exp1 ^ ("<=" ^ exp2)
61         | Geq -> exp1 ^ (">=" ^ exp2)
62         | And ->
63             (wait_for_more exp1 t1) ^
64               ("&&" ^ (wait_for_more exp2 t2))
65         | Or ->
66             (wait_for_more exp1 t1) ^
67               ("||" ^ (wait_for_more exp2 t2))
68         | Send ->
69             "CALL_ENQUEUE_FUNC(" ^
70               (exp1 ^
71                  (", " ^
72                     (exp2 ^
73                        "," ^
```

```
74                          ((translate_type t1) ^ (", " ^ (print_dot ^ ")")))))))
75          | Assign ->
76              (match (t1, (fst typed_exp2)) with
77               | (List t, TListInitializer _) ->
78                   let temp_list_name = "_temp_" ^ exp1 in
79                   let temp_vdecl =
80                     {
81                       s_declaration_type = List t;
82                       s_declaration_id = temp_list_name;
83                       s_declaration_initializer = typed_exp2;
84                     }
85                   in
86                     (translate_vdecl temp_vdecl false) ^
87                       (";\n" ^ (exp1 ^ ("=" ^ (temp_list_name ^ ";\n"))))
88               | _ -> exp1 ^ ("=" ^ exp2))
89          | Concat ->
90              "_add_front( (union _payload) " ^ (exp1 ^ (", " ^ (exp2 ^ ")"))) in
91
92      (* Translate a flow unary operation *)
93      let translate_unary_op (unary_op : unary_op) (typed_expr : typed_expr) : string =
94        let exp = translate_expr typed_expr
95        in
96          match unary_op with
97          | Not -> "!" ^ exp
98          | Negate -> "-" ^ exp
99          | Retrieve ->
100             (match snd typed_expr with
101                  Channel (t, dir) ->
102                  "CALL_DEQUEUE_FUNC(" ^
103                    (exp ^
104                       "," ^
105                          ((translate_type t) ^ (", " ^ (print_dot ^ ")")))))
106             | List t ->
107                 let type_to_union_element =
108                   (function
109                    | Int -> "_int"
110                    | Double -> "_double"
111                    | Char -> "_char"
112                    | Channel (Int, _) -> "_int_channel"
113                    | _ -> "")
114                 in
115                   "_get_front(" ^ (exp ^ (")." ^ (type_to_union_element t)))
116             | _ -> raise (Failure "Invalid type"))
117         | ListLength -> "_get_length(" ^ (exp ^ ")")
118         | ListTail -> "_get_tail(" ^ (exp ^ ")") in
119
120     let translate_bool b = match b with | true -> "1" | false -> "0" in87
121
122     (* Translate a comma separated list of expressions *)
123     let rec translate_expr_list (expr_list : typed_expr list) : string =
124       let translated_exprs =
125         List.rev
126           (List.fold_left (fun acc elm -> (translate_expr elm) :: acc) []
127              expr_list)
128       in String.concat ", " translated_exprs in
129
130     (* Translate flow type functions, including built-ins, to c function calls *)
131     let translate_function_call (id : string) (expr_list : typed_expr list) :
132       string =
133       match id with
134       | "print_string" ->
135           "printf(\"%s\", " ^
136             ((translate_expr_list expr_list) ^ (");\n" ^ "fflush(stdout)"))
137       | "print_int" ->
138           "printf(\"%d\", " ^
139             ((translate_expr_list expr_list) ^ (");\n" ^ "fflush(stdout)"))
140       | "print_char" ->
141           "printf(\"%c\", " ^
142             ((translate_expr_list expr_list) ^ (");\n" ^ "fflush(stdout)"))
143       | "print_double" ->
144           "printf(\"%G\", " ^
145             ((translate_expr_list expr_list) ^ (");\n" ^ "fflush(stdout)"))
```

```
146       | "println" -> "printf(\"\\n\");\n" ^ "fflush(stdout)"
147       | "rand" -> " (double)rand() / (double)RAND_MAX "
148       | _ -> id ^ ("(" ^ ((translate_expr_list expr_list) ^ ")")) in
149
150     (* Translate flow process invocations to c pthread_create's *)
151     let translate_process_call (id : string) (expr_list : typed_expr list) : string =
152       let pthread_decl =
153         "pthread_t* _t = _make_pthread_t(\"" ^ (id ^ "\");\n") in
154       let malloced_args =
155         "struct _" ^
156           (id ^
157              ("_args* _margs = malloc(sizeof(struct _" ^ (id ^ "_args));\n"))) in
158
159       (* Collect the args into a struct on the stack *)
160       let args_struct =
161         "struct _" ^
162           (id ^
163              ("_args _args = {\n" ^
164                 ((translate_expr_list expr_list) ^ "\n};\n"))) in
165
166       (* Copy the struct over to the heap *)
167       let copy_struct =
168         "memcpy((void*) _margs, (void*) &_args, sizeof(typeof(_args)));\n" in
169
170       (* Create the pthread for this process *)
171       let pthread_creation =
172         "pthread_create(_t, NULL, " ^ (id ^ ", (void *) _margs);\n")
173       in
174         "{\n" ^
175           (pthread_decl ^
176              (malloced_args ^
177                 (args_struct ^ (copy_struct ^ (pthread_creation ^ "\n}")))))
178     in
179       match expr with
180       | (TIntLiteral i, _) -> string_of_int i
181       | (TStringLiteral s, _) -> "\"" ^ (s ^ "\"")
182       | (TBoolLiteral b, _) -> translate_bool b
183       | (TCharLiteral c, _) -> "\'" ^ ((String.make 1 c) ^ "\'")
184       | (TDoubleLiteral d, _) -> string_of_float d
185       | (TId i, _) -> i
186       | (TBinOp (expr1, bin_op, expr2), _) ->
187           translate_bin_op expr1 bin_op expr2
188       | (TUnaryOp (unary_op, expr), _) -> translate_unary_op unary_op expr
189       | (TFunctionCall (id, expr_list), Proc) ->
190           translate_process_call id expr_list
191       | (TFunctionCall (id, expr_list), _) -> translate_function_call id expr_list
192       | (TListInitializer expr_list, _) ->
193           "{" ^ ((translate_expr_list expr_list) ^ "}")
194       | (TNoexpr, _) -> ""
195
196   (* Translate flow variable declaration to c variable declaration *)
197   and translate_vdecl (vdecl : s_variable_declaration) (is_arg : bool) =
198     let translated_type = translate_type vdecl.s_declaration_type in
199     translated_type ^
200         (" " ^
201            (vdecl.s_declaration_id ^
202               (" " ^
203                  (* This portion deals with initializing variables. *)
204                  (match vdecl.s_declaration_type with
205                     (* If the declaration is a channel, we need to perform a malloc
206                      * and also initialize the struct associated with the channel *)
207                   | Channel (t, Nodir) ->
208                       (* If channel being translated arg, no malloc needed *)
209                       if is_arg then ""
210                       else
211                         (match fst vdecl.s_declaration_initializer with
212                          | TNoexpr ->
213                              (* Perform the malloc with the proper struct.
214                               * This is taken care of by the runtime with the
215                               * MALLOC_CHANNEL macro *)
216                              "MALLOC_CHANNEL(" ^
217                                ((translate_type t) ^
```

```
218                                    (");\n" ^
219                                       (* This will initializes the locks, flags, etc. *)
220                                       ("_init_channel( (struct _channel *) " ^
221                                          (vdecl.s_declaration_id ^ ")"))))
222
223                              (* Scenario where a channel is dequeued from a list
224                               * or returned from a function *)
225                             | TUnaryOp (Retrieve, _) | TFunctionCall (_, _) ->
226                                 " = " ^
227                                   (translate_expr
228                                      vdecl.s_declaration_initializer)
229                             | _ -> "")
230
231                      | List t ->
232                          (* If the list is an arg, it need not be initialized *)
233                          if is_arg then ""
234                          else
235                            (let list_initialization_statements =
236                               (* Lists can be initialized in a number of ways, ranging
237                                * from intialization lists to function calls, to assignment. *)
238                               match fst vdecl.s_declaration_initializer with
239                               | TListInitializer expr_list ->
240                                   List.map
241                                     (* Call _add_front on every expression in the initializer *)
242                                     (fun expr ->
243                                        vdecl.s_declaration_id ^
244                                          (" = _add_front( (union _payload)" ^
245                                             ((translate_expr expr) ^
246                                                ("," ^
247                                                   (vdecl.s_declaration_id ^ ")")))))
248                                     (List.rev expr_list)
249                               | TUnaryOp (ListTail, _) ->
250                                   [ vdecl.s_declaration_id ^
251                                       ("=" ^
252                                          (translate_expr
253                                             vdecl.s_declaration_initializer)) ]
254                               | TId id_name ->
255                                   [ vdecl.s_declaration_id ^ ("=" ^ id_name);
256                                     "_increase_refs(" ^ (id_name ^ ")") ]
257                               | TFunctionCall (_, _) ->
258                                   [ vdecl.s_declaration_id ^
259                                       ("=" ^
260                                          (translate_expr
261                                             vdecl.s_declaration_initializer)) ]
262                               | TNoexpr -> [ "" ]
263                               | _ -> raise (Failure "Invalid list initializer ")
264                             in "= NULL; " ^ (String.concat ";\n" list_initialization_statements))
265                      | _ ->
266                          (match vdecl.s_declaration_initializer with
267                           | (TNoexpr, _) -> ""
268                           | (_, _) ->
269                               "=" ^
270                                 (translate_expr vdecl.s_declaration_initializer))))) in
271
272   (* Translates specifically those expressions that are used in a
273    * boolean context. This is necessary to check if channel used as a
274    * boolean. *)
275   let translate_boolean_expr (typed_expr : typed_expr) : string =
276     let t = snd typed_expr
277     in
278       match t with
279       | Channel (_, _) ->
280           "_wait_for_more((struct _channel* ) " ^
281             ((translate_expr typed_expr) ^ ")")
282       | _ -> translate_expr typed_expr in
283
284   (* Check the type of the poison token *)
285   let translate_poison_expr (typed_expr : typed_expr) : string =
286     let t = snd typed_expr
287     in
288       match t with
289       | Channel (_, _) ->
```

```
290           "_poison((struct _channel* )" ^
291             ((translate_expr typed_expr) ^ ");")
292       | _ -> translate_expr typed_expr in
293
294   (* Translate a flow statement to a C statement *)
295   let rec translate_stmt (stmt : s_stmt) : string =
296     match stmt with
297     | SExpr e ->
298         let translated_expr = (translate_expr e) ^ ";\n"
299         in
300           (match fst e with
301            | (* This absurd match finds all list assignments for ref counting *)
302                TBinOp (e1, op, e2) when (op = Assign) &&
303                 (match snd e1 with | List _ -> true | _ -> false) ->
304                     let list_name = translate_expr e1 in
305                     let store_temp = "temp = " ^ (list_name ^ ";\n") in
306                     let dec_stmt = "_decrease_refs(temp);\n"
307                     and inc_stmt = "_increase_refs(" ^ (list_name ^ ");\n") in
308                     (* First decrease the references to the list, in
309                      * prep for reassignment. Then, do the reassignment.
310                      * Then, increase the references to the list, which
311                      * now points to a new cell after reassignment *)
312                  store_temp ^ (translated_expr ^ (inc_stmt ^ dec_stmt))
313            | _ -> translated_expr)
314     | SBlock stmt_list ->
315         "{\n" ^
316           ((String.concat "" (List.map translate_stmt stmt_list)) ^ "}\n")
317     | SReturn e -> "return " ^ ((translate_expr e) ^ ";")
318     | SDeclaration vdecl -> (translate_vdecl vdecl false) ^ ";\n"
319     | SIf (e1, s1, s2) ->
320         "if(" ^
321           ((translate_boolean_expr e1) ^
322              (")\n" ^
323                 ((translate_stmt s1) ^ ("\nelse\n" ^ (translate_stmt s2)))))
324     | SFor (e1, e2, e3, s) ->
325         "for(" ^
326           ((String.concat "; " (List.map translate_expr [ e1; e2; e3 ])) ^
327              (")\n" ^ (translate_stmt s)))
328     | SWhile (e, s) ->
329         "while(" ^ ((translate_boolean_expr e) ^ (")" ^ (translate_stmt s)))
330     | SContinue -> "continue;"
331     | SBreak -> "break;"
332     | SPoison e -> translate_poison_expr e
333     | SExitProc -> "_exit_thread();" in
334
335   (* unpacks the arguments to a process from void *_args *)
336   let unpack_process_args (process : s_function_declaration) : string =
337     "\n" ^
338       ((String.concat ";\n"
339           (List.map
340              (fun vdecl ->
341                 (translate_vdecl vdecl true) ^
342                   (" = " ^
343                      ("((struct _" ^
344                         (process.s_function_name ^
345                            ("_args*) _args)->" ^ vdecl.s_declaration_id)))))
346              process.s_arguments))
347         ^ ";\n")
348   in
349
350   (* Translate flow function declaration to c function declaration *)
351   let translate_fdecl (fdecl : s_function_declaration) : string =
352
353     (* Opening and closing statments are required to in main to initialize
354      * and clean up the environment, respectively *)
355     let (opening_stmts, closing_stmts) =
356       if fdecl.s_function_name = "main"
357       then
358         (("_initialize_runtime(" ^ (print_dot ^ ");\n")),
359          ("_wait_for_finish(" ^ (print_dot ^ ");\n")))
360       else ("", "")
361
```

```
362     (* temp is a helper variable present in every function. It's used to
363      * juggle around temporary lists during list reassignment for the
364      * purpose of reference counting. *)
365     and temp_list_decl = "struct _cell* temp;\n" in
366
367     (* Translate the function's argument declarations *)
368     let arg_decl_string_list =
369       List.map (fun arg -> translate_vdecl arg true) fdecl.s_arguments
370     in
371       (* Procs must have their args bundled into a struct. So, during the
372        * declaration of a proc, that proc's argument struct must be declared. *)
373       (match fdecl.s_return_type with
374        | Proc ->
375            "struct _" ^
376              (fdecl.s_function_name ^
377                 ("_args{\n\t" ^
378                    ((String.concat ";\n\t" arg_decl_string_list) ^ ";\n};\n")))
379        | _ -> "") ^
380         (* This is the actual c function declaration *)
381         ((translate_type fdecl.s_return_type) ^
382            (" " ^
383               (fdecl.s_function_name ^
384                  ((match fdecl.s_return_type with
385                    | Proc -> "(void *_args)\n{" ^ (unpack_process_args fdecl)
386                    | _ ->
387                        "(" ^
388                          ((String.concat ", " arg_decl_string_list) ^ ")\n{")) ^
389                   (opening_stmts ^
390                      (temp_list_decl ^
391                         ((String.concat ""
392                             (List.map translate_stmt fdecl.s_body))
393                           ^ (closing_stmts ^ "\n}")))))))))
394   in
395
396     (* Translate the flow program to a c program *)
397     boilerplate_header ^       (* The base c runtime environment *)
398       ((String.concat "\n"     (* This translates all top-level function and variable decls. *)
399           (List.map
400              (fun decl ->
401                 match decl with
402                 | SVarDecl vdecl -> (translate_vdecl vdecl false) ^ ";\n"
403                 | SFuncDecl fdecl -> translate_fdecl fdecl)
404              (List.rev program)))
405          ^ "\n")
```

8.7 c_runtime.c

```
 1 /* c_runtime.c */
 2 #include <assert.h>
 3 #include <pthread.h>
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6 #include <stdbool.h>
 7 #include <string.h>
 8 #include <time.h>
 9
10 /***************************** Channel Structs ******************************/
11 #define BASIC_CHANNEL_MEMBERS                                                \
12   pthread_mutex_t lock;                                                      \
13   int size;                                                                  \
14   bool poisoned;                                                             \
15   pthread_cond_t write_ready;                                                \
16   pthread_cond_t read_ready;                                                 \
17   int front;                                                                 \
18   int back;                                                                  \
19   int MAX_SIZE;                                                              \
```

```
20   int claimed_for_writing;                                                \
21   int claimed_for_reading;                                                \
22   pthread_t writing_thread;                                               \
23   pthread_t reading_thread;
24
25 struct _channel {
26   BASIC_CHANNEL_MEMBERS
27 };
28
29 struct _int_channel {
30   BASIC_CHANNEL_MEMBERS
31   int queue[100];
32 };
33
34 struct _char_channel {
35   BASIC_CHANNEL_MEMBERS
36   char queue[100];
37 };
38
39 struct _double_channel {
40   BASIC_CHANNEL_MEMBERS
41   double queue[100];
42 };
43
44 #define MALLOC_CHANNEL(type)                                              \
45   = (struct _##type##_channel *)malloc(sizeof(struct _##type##_channel));
46
47 int _init_channel(struct _channel *channel) {
48   if (pthread_mutex_init(&channel->lock, NULL) != 0) {
49     printf("Mutex init failed");
50     return 1;
51   }
52
53   if (pthread_cond_init(&channel->write_ready, NULL) +
54           pthread_cond_init(&channel->read_ready, NULL) !=
55       0) {
56     printf("Cond init failed");
57     return 1;
58   }
59   channel->claimed_for_reading = 0;
60   channel->claimed_for_writing = 0;
61   channel->MAX_SIZE = 100;
62   channel->front = 0;
63   channel->back = 0;
64   channel->poisoned = false;
65   return 0;
66 }
67
68 /************************ Global Thread Metadata *************************/
69
70 /* Node for linked list of channel names. Keeps track of channels that
71  * threads can write to. */
72 struct _channel_list_node {
73   struct _channel *chan;
74   struct _channel_list_node *next;
75 };
76
77 /* Defines a node of the global thread metadata list. */
78 struct _pthread_node {
79   pthread_t thread;
80   struct _pthread_node *next;
81   char *proc_name;
82   struct _channel_list_node *writing_channels;
83 };
84
85 /* The global thread metadata list */
86 struct _pthread_node *_head = NULL;
87 struct _pthread_node *_tail = NULL;
88
89 /* Lock for the thread metadata list */
90 pthread_mutex_t _thread_list_lock;
91 pthread_mutex_t _ref_counting_lock;
```

```
 92
 93 /* Finds a thread in the global threadlist given its id */
 94 struct _pthread_node *_get_thread(pthread_t thread_id) {
 95   pthread_mutex_lock(&_thread_list_lock);
 96   struct _pthread_node *curr = _head;
 97   while (curr) {
 98     if (curr->thread == thread_id) {
 99       break;
100     }
101     curr = curr->next;
102   }
103   pthread_mutex_unlock(&_thread_list_lock);
104   return curr;
105 }
106
107 /* Gets the name of the process running on a thread,
108  * given the thread id */
109 char *_get_thread_name(pthread_t thread_id) {
110   if (_head == NULL)
111     return "";
112   pthread_mutex_lock(&_thread_list_lock);
113   char *name = "";
114   struct _pthread_node *curr = _head;
115   while (curr) {
116     if (curr->thread == thread_id) {
117       name = curr->proc_name;
118       break;
119     }
120     curr = curr->next;
121   }
122   pthread_mutex_unlock(&_thread_list_lock);
123   return name;
124 }
125
126 void _print_dot_node(struct _channel *chan) {
127   fprintf(stderr, "{%d[label=%s]}->{%d[label=%s]}\n", (int)chan->writing_thread,
128           _get_thread_name(chan->writing_thread), (int)chan->reading_thread,
129           _get_thread_name(chan->reading_thread));
130 }
131
132 /************************ Enqueue/Dequeue Macros ****************************/
133
134 /* Given a token type, this macro generates an enqueue function
135  * for the associated channel. */
136 #define MAKE_ENQUEUE_FUNC(type)                                              \
137   type _enqueue_##type(type element, struct _##type##_channel *channel,      \
138                        bool dot_print) {                                     \
139     pthread_mutex_lock(&channel->lock);                                      \
140     pthread_t this_thread = pthread_self();                                  \
141     if (!channel->claimed_for_writing) {                                     \
142       channel->claimed_for_writing = 1;                                      \
143       channel->writing_thread = this_thread;                                 \
144       struct _pthread_node *this_thread_node = _get_thread(this_thread);     \
145       struct _channel_list_node *new_writing_chan =                          \
146           malloc(sizeof(struct _channel_list_node));                         \
147       new_writing_chan->next = this_thread_node->writing_channels;           \
148       new_writing_chan->chan = (struct _channel *)channel;                   \
149       this_thread_node->writing_channels = new_writing_chan;                 \
150       if (channel->claimed_for_reading && dot_print)                         \
151         _print_dot_node((struct _channel *)channel);                         \
152     } else if (channel->writing_thread != this_thread) {                     \
153       fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "   \
154                       "write to a channel belonging to %s (thread 0x%x)\n",  \
155               _get_thread_name(this_thread), (int)this_thread,               \
156               _get_thread_name(channel->writing_thread),                     \
157               (int)channel->writing_thread);                                 \
158       exit(1);                                                               \
159     }                                                                        \
160     while (channel->size >= channel->MAX_SIZE)                               \
161       pthread_cond_wait(&channel->write_ready, &channel->lock);              \
162     assert(channel->size < channel->MAX_SIZE);                               \
163     if (channel->poisoned) {                                                 \
```

```
164       fprintf(stderr,                                                          \
165               "Attempting to read from a channel that is empty and poisoned"); \
166       exit(1);                                                                 \
167     }                                                                          \
168     channel->queue[channel->back] = element;                                   \
169     channel->back = (channel->back + 1) % channel->MAX_SIZE;                   \
170     channel->size++;                                                           \
171     pthread_cond_signal(&channel->read_ready);                                 \
172     pthread_mutex_unlock(&channel->lock);                                      \
173     return element;                                                            \
174   }
175
176 /* Create enqueue functions for ints, chars, and doubles */
177 MAKE_ENQUEUE_FUNC(int)
178 MAKE_ENQUEUE_FUNC(char)
179 MAKE_ENQUEUE_FUNC(double)
180
181 /* This macro calls the appropriate dequeue function */
182 #define CALL_ENQUEUE_FUNC(e, c, t, dot) _enqueue_##t(e, c, dot)
183
184 /* Given a token type, this macro generates a dequeue function
185  * for the associated channel. */
186 #define MAKE_DEQUEUE_FUNC(type)                                                \
187   type _dequeue_##type(struct _##type##_channel *channel, bool dot_print) {    \
188     pthread_mutex_lock(&channel->lock);                                        \
189     pthread_t this_thread = pthread_self();                                    \
190     if (!channel->claimed_for_reading) {                                       \
191       channel->claimed_for_reading = 1;                                        \
192       channel->reading_thread = this_thread;                                   \
193       if (channel->claimed_for_writing && dot_print)                           \
194         _print_dot_node((struct _channel *)channel);                           \
195     } else if (channel->reading_thread != this_thread) {                       \
196       fprintf(stderr, "Runtime error: proc %s (thread 0x%x) is trying to "     \
197                       "read from a channel belonging to %s (thread 0x%x)\n",   \
198               _get_thread_name(this_thread), (int)this_thread,                 \
199               _get_thread_name(channel->reading_thread),                       \
200               (int)channel->reading_thread);                                   \
201       exit(1);                                                                 \
202     }                                                                          \
203     if (channel->size == 0) {                                                  \
204       fprintf(stderr, "Attempting to read from empty channel");                \
205       exit(1);                                                                 \
206     }                                                                          \
207     type result = channel->queue[channel->front];                              \
208     channel->front = (channel->front + 1) % channel->MAX_SIZE;                 \
209     channel->size--;                                                           \
210     pthread_cond_signal(&channel->write_ready);                                \
211     pthread_mutex_unlock(&channel->lock);                                      \
212     return result;                                                             \
213   }
214
215 /* Make dequeue functions for int, char, and double channels */
216 MAKE_DEQUEUE_FUNC(int)
217 MAKE_DEQUEUE_FUNC(char)
218 MAKE_DEQUEUE_FUNC(double)
219
220 /* This macro calls the appropriate dequeue function */
221 #define CALL_DEQUEUE_FUNC(c, t, dot) _dequeue_##t(c, dot)
222
223 /* Poison the channel, indicating that it won't be written to in the future */
224 void _poison(struct _channel *channel) {
225   pthread_mutex_lock(&channel->lock);
226   channel->poisoned = true;
227   pthread_cond_signal(&channel->read_ready);
228   pthread_mutex_unlock(&channel->lock);
229 }
230
231 /* This function is called whenever a channel is used in a boolean context.
232  * Three cases:
233  *   1) Channel is poisoned and empty     -> return false
234  *   2) Channel is nonempty               -> return true
235  *   3) Channel is empty but not poisoned -> block
```

```
236  */
237 bool _wait_for_more(struct _channel *channel) {
238   pthread_mutex_lock(&channel->lock);
239   while (channel->size == 0) {
240     if (channel->poisoned) {
241       pthread_mutex_unlock(&channel->lock);
242       return false;
243     } else {
244       pthread_cond_wait(&channel->read_ready, &channel->lock);
245     }
246   }
247   pthread_mutex_unlock(&channel->lock);
248   return true;
249 }
250
251 /************************ Miscellaneous ************************/
252
253 /* Initializes global locks */
254 void _initialize_runtime(bool print_dot) {
255   pthread_mutex_init(&_thread_list_lock, NULL);
256   pthread_mutex_init(&_ref_counting_lock, NULL);
257   srand(time(NULL));
258   if (print_dot)
259     fprintf(stderr, "digraph G{\n");
260 }
261
262 /* Create a pthread node and enqueue it on the list. Return the address of
263  * its id for pthread_create */
264 pthread_t *_make_pthread_t(char *proc_name) {
265   pthread_mutex_lock(&_thread_list_lock);
266   struct _pthread_node *new_pthread =
267       (struct _pthread_node *)malloc(sizeof(struct _pthread_node));
268   new_pthread->next = NULL;
269   new_pthread->proc_name = proc_name;
270   new_pthread->writing_channels = NULL;
271   if (_head == NULL) {
272     _head = _tail = new_pthread;
273   } else {
274     _tail->next = new_pthread;
275     _tail = new_pthread;
276   }
277   pthread_mutex_unlock(&_thread_list_lock);
278   return &(new_pthread->thread);
279 }
280
281 /* Invoked when return is reached from a process.
282  * This function will cause the returning thread to poison all of
283  * its outgoing channels if it hasn't done so yet. */
284 void _exit_thread() {
285   struct _pthread_node *this_thread = _get_thread(pthread_self());
286   struct _channel_list_node *curr_chan = this_thread->writing_channels;
287   while (curr_chan) {
288     if (!curr_chan->chan->poisoned)
289       _poison(curr_chan->chan);
290     curr_chan = curr_chan->next;
291   }
292   pthread_exit(NULL);
293 }
294
295 /* Called from within main to wait for processes to finish */
296 void _wait_for_finish(bool print_dot) {
297   struct _pthread_node *curr = _head;
298   while (curr) {
299     pthread_join(curr->thread, NULL);
300     curr = curr->next;
301   }
302   if (print_dot)
303     fprintf(stderr, "}");
304 }
305
306 /****************************** Lists ******************************/
307 union _payload {
```

```
308   int _int;
309   double _double;
310   char _char;
311   void *_cell;
312   struct _int_channel *_int_channel;
313 };
314
315 struct _cell {
316   struct _cell *next;
317   union _payload data;
318   int references;
319   int length;
320 };
321
322 struct _cell *_add_front(union _payload element, struct _cell *tail) {
323   struct _cell *new_cell = malloc(sizeof(struct _cell));
324   new_cell->references = 1;
325   new_cell->data = element;
326   new_cell->next = tail;
327   if (!tail)
328     new_cell->length = 1;
329   else {
330     new_cell->length = tail->length + 1;
331     tail->references++;
332   }
333   return new_cell;
334 }
335
336 struct _cell *_get_tail(struct _cell *head) {
337   if (!head) {
338     fprintf(stderr, "Runtime error: cannot get tail of empty list");
339     exit(1);
340   }
341
342   return head->next;
343 }
344
345 void __decrease_refs(struct _cell *head, int lock) {
346   if(lock)
347     pthread_mutex_lock(&_ref_counting_lock);
348
349   if(!head){
350     if(lock)
351       pthread_mutex_unlock(&_ref_counting_lock);
352     return;
353   }
354   else if(head->references > 1)
355     head->references--;
356   else{
357     __decrease_refs(head->next, 0);
358     free(head);
359   }
360   if(lock)
361     pthread_mutex_unlock(&_ref_counting_lock);
362 }
363
364 void _decrease_refs(struct _cell *head) {
365   //__decrease_refs(head, 1);
366 }
367
368 void _increase_refs(struct _cell *head) {
369   pthread_mutex_lock(&_ref_counting_lock);
370   if (head)
371     head->references++;
372   pthread_mutex_unlock(&_ref_counting_lock);
373 }
374
375 union _payload _get_front(struct _cell *head) {
376   if (!head) {
377     fprintf(stderr, "Runtime error: cannot get head of empty list");
378     exit(1);
379   }
```

```
380
381   return head->data;
382 }
383
384 int _get_length(struct _cell *head) {
385   if (!head)
386     return 0;
387   return head->length;
388 }
```

8.8 flowc.ml

Authors:

```
 1 (* flowc.ml *)
 2 open Ast
 3 open Sast
 4
 5 type action = | Ast | Sast | Compile | Dot
 6
 7 let _ =
 8   let (action, file) =
 9     if (Array.length Sys.argv) > 2
10     then
11       ((List.assoc Sys.argv.(1)
12           [ ("-a", Ast); ("-c", Compile); ("-d", Dot); ("-s", Sast) ]),
13        (open_in Sys.argv.(2)))
14     else (Compile, (open_in Sys.argv.(1))) in
15   let lexbuf = Lexing.from_channel file in
16   let program = Parser.program Scanner.token lexbuf in
17   let sprogram =
18     try Semantic_analysis.check_progam program
19     with
20     | Failure m ->
21         (prerr_endline ("Error in semantic analysis\n" ^ m);
22          flush stdout;
23          exit 1)
24   in
25     match action with
26     | Ast ->
27         ignore
28           (let _ = Printer.print_string_of_program program in
29            let graph = "digraph G{" ^ (!Printer.dot_graph ^ "}") in
30            let outfile = open_out "out.dot" in
31            let _ = Printf.fprintf outfile "%s" graph in
32            let _ = close_out outfile
33            in Sys.command "dot -Tpng out.dot -o out.png")
34     | Sast ->
35         ignore
36           (let _ = Sprinter.print_string_of_program sprogram in
37            let graph = "digraph G{" ^ (!Sprinter.dot_graph ^ "}") in
38            let outfile = open_out "out.dot" in
39            let _ = Printf.fprintf outfile "%s" graph in
40            let _ = close_out outfile
41            in Sys.command "dot -Tpng out.dot -o out.png")
42     | Compile -> print_string (Compile.compile sprogram false)
43     | Dot -> print_string (Compile.compile sprogram true)
```