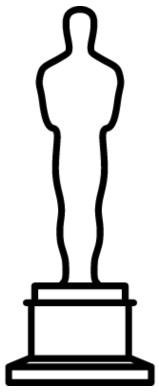
Oscar

A Functional, Actor based Programming Language



Ethan Adams (ea2678)

Howon Byun (hb2458)

Jibben Hillen (jlh2218)

Vladislav Scherbich (vss2113)

Anthony Holley (ajh2186)

Contents

1 Introduction

2 Language Tutorial

2.1 Using the Compiler

2.2 Core Concepts

2.2.1 Immutability

2.2.2 Actors

2.3 Example Program

3 Language Reference Manual

3.1 Lexical Conventions

3.2 Identifiers

3.2.1 Whitespace

3.3 Keywords

3.5 Punctuation

3.6 Types

3.6.1 Primitive Types

3.6.2 Non-Primitive Types

3.7 Program Structure

3.7.1 Logic and Relations

3.7.2 Control Flow

3.7.3 Actor, Pool & Message

3.7.4 Executable

3.8 Built-in Functions

3.8.1 The Println Function

3.8.2 Type Conversion Functions

3.8.3 Container Specific Functions

3.9 Scoping and Immutability

3.10 Standard Library

4 Project Plan

4.1 Planning Process

4.2 Specification Process

4.3 Development Process

4.4 Testing Process

4.5 Team Responsibilities

4.6 Software Development Environment

4.7 Programming Style Guide

4.8 Project Timeline

4.9 Project Log

5 Architecture

5.1 The Compiler

5.1.1 Scanner

5.1.2 Parser

5.1.3 "Pretty Printer"

5.1.4 Semantic Analyzer

5.1.5 Code Generator

5.1.6 Optimizer

5.2 The C++ Libraries

6 Test Plan

6.1 Testing Phases

6.1.1 Unit Testing

6.1.2 Integration Testing

6.2 Automation

6.3 Test Suites

7 Team Experience

7.1 *Howon*

7.2 *Jibben*

7.3 *Ethan*

7.4 *Vlad*

7.5 *Anthony*

8 Code Listing

8.1 Oscar Files

8.2 C++ Backend

8.3 Test Suite Files

9 Git Log

1. Introduction

Oscar is an an Actor-Oriented Language. As all devs know, there can be difficulties and frustrations associated with parallel computation, namely data collision when multiple threads attempt to access the same data. When handling threads directly, resolving this issue most often requires explicitly defining locks and mutexes to control data access, which adds significant mental and code overhead. The Actor model is a different approach to this problem that relies on the concept of message-passing between processes. The model simplifies the aforementioned difficulties of multiprocessing by abstracting explicit method calls, and relying on each “actor” to handle tasks and encapsulate mutable data. Oscar is based off the Erlang and Akka framework.

2. Language Tutorial

2.1 Using the Compiler

In the *Oscar* directory, type **make**. This will create the Oscar compiler, `oscar`. This is an executable that can be run as such:

```
$ ./oscar [-p|-s|-l|-c] [?-O] <oscar file>
```

Where `<oscar file>` is the path to your `.oscar` file.

There are several flags that you can specify to carry out different operations on your `.oscar` file.

- ❏ The `-p` flag signals for the compiler to generate an abstract syntax tree and pretty print your program. This will run your program through tokenization and parsing, and print out the generated AST. Pretty printing will print the AST such that it will match the text of your program exactly.
- ❏ The `-s` flag signals for the compiler to generate a syntactically checked abstract syntax tree and pretty print your program. This is similar to the above `-p` flag, except that your program is also semantically checked. Again, the printed result of the SAST will match the text of your program exactly.
- ❏ The `-l` flag signals for the compiler to generate C++ code and LLVM IR. This will generate `.cpp` and `.ll` files that matches the name of your `.oscar` file.
- ❏ The `-c` flag signals for the compiler to generate C++ code and an executable file, both of which will match the name of your `.oscar` file.

  For example, the the following command will generate the C++ file `helloworld.cpp` and the executable `helloworld`.

  ```
  $ ./oscar -c helloworld.oscar
  ```

- ❏ The optional `-O` flag signals the compiler to optimize your code. For additional information about optimization, please look into **section 5.1.6** on the optimizer.

  For example, the the following command will generate the optimized C++ file `helloworld.cpp` and the executable `helloworld`.

  ```
  $ ./oscar -c -O helloworld.oscar
  ```

Note: To be able to use Oscar's library functions effectively, the Makefile will copy Oscar's C++ bindings to the directory `/usr/local/include/oscar/` . This is the standard location for C++ libraries. It is possible that you may run into permissions issues, depending on your environment.

2.2 Core Concepts

2.2.1 Immutability

Values in Oscar are immutable like Haskell and Ocaml. This means that when a non-atomic value is declared, it cannot be reassigned and function parameters are passed by value. For instance, this is not allowed.

```
int x = 5;
x = 4; // error
```

For primitive container types like `list, map` and `set`, they have an additional property that re-assigning values returns a new copy of the data structure with those updates applied.

```
list<int> initList == list<int>[1, 2, 3, 4, 5];
list<int> changedList = (initList[3] = -4);
changedList == [1, 2, 3, -4, 5];
initList == [1, 2, 3, 4, 5];
```

Passing objects by value seems like a massive performance drain. After all, if a large container, say a list of five million entries, were passed in, then there is a huge memory overhead to copy such structure. `List`, and `map` and `set` that are implemented by it, in Oscar are implemented under the hood as C++ shared pointers with mutability checked in the semantic analyzer so that collection manipulation is as efficient as possible

The main reason for implementing this is to enable painless parallel operations. Compared to traditional imperative languages like C++ and Java where variables are mutable by default and explicit synchronization methods are required to prevent data races, Oscar's way of delegating actors to handle these mean that the user does not have to worry about synchronization primitives. Actors are discussed in further detail below.

With this said mutability is allowed within one context: actors. When variables are declared with `mut` keyword, reassignments are allowed. Non-container primitives can have their values changed and container types allow reassignments of the values held.

```
 actor {
   mut int x = 10 /* allowed within actors */
   x = 14 / x == 14 now since x was declared to be mutable */
   mut list<int> mutableList = list<int>[1, 2, 3, -4, 5]
   list<int> alteredList = (mutableList[3] = 5);
   mutableList == alteredList == [1, 2, 3, 5, 5]
}
```

2.2.2 Actors

The **Actor** is the basic unit of parallel processing in our language. A one sentence startup pitch for actors would be like a Java class that extends Runnable interface where all of the fields are private and the only way to affect it is through passing a specialized construct called **messages**.

The most often used method of achieving concurrency is threading. Threads allow a process to efficiently perform multiple tasks at once (given necessary hardware/OS). Issue with most imperative languages is that, as mentioned above, data races must be handled explicitly, which means synchronization methods like locks and semaphores are needed that sacrifices performance.

Actors fix this issue by preventing data races through encapsulation of data within its own unit of processing and strict enforcing of means of communication among them.[1] As mentioned before, fields declared inside actors are private, meaning other actors and programs cannot access and change them. This prevents side effect mutations that often cause aforementioned problems. As such, the only means of affecting other actors is through message passing. Each actor defines the types of message it can process and corresponding action it will take based on that message. By doing so, the concern of data races are eliminated- one actor simply cannot implicitly modify states of others.
This is why mutable variables allowed within actors. Since these variables cannot be modified by outside functions or actors unless message handles are specifically implemented, these mutations can happen safely without worrying about data race conditions. Oscar elevates these concepts as primitives types as well as multiple operations to enable inter-actor communications.

2.3 Example Program

Here is a sample Oscar to approximate the value of pi that incorporate these features.

```
message start() /* empty message */
message end()
message work(start: int, numElems: int)
message result(value: double)
message piApproximation(pi: double)

actor Worker {
  def calcPi(start: int, numElems: int) => double = {
    return [i <- start to (start + numElems)].map(i => {
      return 4.0 * (1 - (i % 2) * 2) / (2 * i + 1)
    }).reduce((x, y) => x + y) /* list operations */
  }

  receive {
    | work(start: int, numElems: int) => {
```

[1] https://en.wikipedia.org/wiki/Actor_model

```
        message<result>(calcPi(start, numElems)) |> sender;
      }
    }
}

actor Listener(name: string) {
  receive {
    | piApproximation(value: double) => {
        Println("value of pi is approximately :" + value);
        message<end>() |> sender;
      }
  }
}

/* Master worker for pi approximation */
actor Master(numWorkers: int, numMsgs: int, numElems: int) {
  mut pi = 0.0;
  mut numResults = 0;
  mut pool<Worker> workerPool = spawn pool<Worker>(numWorkers);
  actor<Listener> listener = spawn actor<Listener>("pi listener");

  receive {
    | start() => {
        list<work> msgList = [int i <- 0 to numMsgs by 1].map(i => {
          work(i * numElems, numElems);
        })
         msgList |>> workerPool;
      }
    | result() => {
        pi = pi + value;
        numResults = numResults + 1;
        if (numResults == numMsgs) {
          piApproximation(pi) |> listener;
        }
      }
    | end() => {
        Die();
      }
  }
}

def main() => unit = { /* main is a keyword */
  actor<Master> master = spawn actor<Master>(5, 10000, 10000);
  message<start>() |> master;
}
```

3. Language Reference Manual

3.1 Lexical Conventions

One of the main goals of Oscar is to be consistent and clear. In languages like Java, there are way too many syntactic sugars that lend itself to inconsistent coding conventions. For example, `array`s contain value of `length` yet `ArrayList`s rely on `size()`. In Oscar, all contains use the same convention of initializing with `[value1, value2, ...]` to maintain consistency and ease of development by preventing developers from having to lookup different APIs for achieving the same goal.

3.2 Identifiers

Identifiers consist of ASCII characters that must start with a letter then followed any combination of letters and numbers. In other words, it must follow this regex

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

3.2.1 Whitespace

Any combination of tabs, spaces and new lines are allowed. Oscar is not whitespace sensitive. Semicolons are used for statement separation

3.3 Keywords

Keyword	Usage	Example

`mut`	Declaration for mutable data
	Inside actors only
		`mut int i = 5;`
		`i = 4;`

`actor`	Actor as a first class object	`actor Worker {}`

`die`	Kills the actor and all actors spawned by this actor	`receive = {`
		`  | end() => { die(); }`
		`}`

`spawn`	Used to spawn actors and pools	`spawn Worker("worker");`
		`spawn pool<Worker>({"worker pool"}, 10);`

`pool`	Construct used to manage a collection of workers	`pool<Worker> p = spawn pool<Worker>({"worker pool"}, 10);`

`receive`	Message handler in actor	`receive {`
		`  | messageType(arg: type) => { ... }`
		`}`

`sender`	Source worker of the message	`message<Say>("hi") |> sender;`

`message`	Signal between actors and pools	`message<Say>("hi") |> sender;`
`return`	Return	`def f(x: int) => unit = { return x; }`
`if/else`	Conditionals	`if (true) {} else {}`
`int`	Integer data type	`int x = 4;`
`double`	Double data type used for floating point arithmetics	`double d = 4.4;`
`char`	Characters	`char c = 'c';`
`bool`	Boolean data type. Lowercase true/false	`bool lie = true;`
`string`	String	`string str = "hi";`
`list`	List backed with HAMT[2] for persistence with performance.	`list<int> intList = list<int>[1, 2, 3];`
`set`	Set of types	`set<int> intSet = set<int>[1, 2, 3];`
`map`	Map of key value pairs	`map<int, int> map = map<int, int>[1 -> 2, 3 -> 4];`
`def/func`	Functions	`def f(x: int) => int = { return x };`
`main`	Main function	`def main(args: list<string>) => unit = { ... }`

3.4 Punctuation

Punctuation	Usage	Example
`.`	Decimal part indicator	`42.17`
`,`	Container separator	`list<int> a = [1, 2, 3, 4, 5];`
`;`	Statement separator	`int x = 4; int j = 4;`
`[]`	List/set/map declaration	`list<int> a = [1, 2, 3, 4, 5];`
`[]`	List/set/map access	`a[1]; // 1`
`{}`	Statement block	`if (true) { println("hi"); }`
`()`	Conditional delimiter	`if (false) { }`
`()`	Function arguments declaration	`def func(arg: type) => unit = { ... }`
`''`	Character literal	`char c = 'a';`

[2] http://lampwww.epfl.ch/papers/idealhashtrees.pdf

`“”`	String literal	`String str = “hi”;`

`<>`	Complex type declaration	`list<string> strList = [“hi”];`
		`actor<Worker> w = spawn actor<Worker>();`
		`map<int, char> = [1 -> ‘a’];`

`/* */`	Multi-line comment	`/* hi`
		` * hello`
		` */`

3.5 Operators

Operator	Usage	Associativity	Types allowed
`+`	Addition	Left	Int, double, string
`–`	Subtraction	Left	Int, double
`*`	Multiplication	Left	Int, double
`/`	Division	Left	Int, double
`%`	Modulo	Left	Int
`=`	Assignment	Right	any
`==`	Equal to	None	Int, double, string, char, unit, list, set, map
`!`	Logical negation	Right	bool
`!=`	Not equal to	None	Int, double, string, char, unit, list, set, map
`>`	Greater than	None	Int, double
`>=`	Greater than or equal to	None	Int, double
`<`	Less than	None	Int, double
`<=`	Less than or equal to	None	Int, double
`&&`	Logical AND	Left	bool
`||`	Logical OR	Left	bool
`&`	Bitwise AND	Right	Int
`|`	Bitwise OR	Right	Int

^	Bitwise XOR	Right	Int
<<	Bitwise left shift	Right	Int
>>	Bitwise right shift	Right	Int
()	Function invocation	Right	
:	Function argument type declaration	Left	
=>	Function return type declaration	Left	
->	Map key-value pair	Left	any - any
|>	Send a message to an actor	Left	message - actor
|>>	Broadcast a message to a pool	Left	message - pool

Precedence as would be in ocamlyacc

```
,           <---  low
=
: -> => <-
&& ||
& ^ |
< <= > >= == !=
<< >>
+ -
* / %
~
|> |>>    <---     high
```

3.6 Types

By default, everything is immutable in Oscar. This means that values cannot be re-assigned and container values return a new copy of the container when modified.
Mutable values are allowed in one instance: as field variables of actors. This allows explicit, rather than implicit, changes of values through message passing. Note: This section includes several examples of Oscar's built-in functions for type conversion and interfacing with containers. Additional documentation on these methods in provided in **section 3.8.**

3.6.1 Primitive Types

Keyword	Usage	Example
`int`	Integer data type	`int x = 4;`
`double`	Double data type used for floating point arithmetics	`double d = 4.4;`
`char`	Characters	`char c = ‘c’;`

`bool`	Boolean data type. Lowercase true/false	`bool t = true;`
		`bool f = false;`

`unit`	Unit	`def nothing() => () = println();`

3.6.1.1 Integer

An integer type is an immutable, signed 4 byte value of digits

```
int x = 5;
x = 4; /* ERROR */
```

Integers can be casted into doubles using “AsDouble()” method.

```
int x = 5;
X == 5.; /* ERROR */
AsDouble(x) == 5.;
```

3.6.1.2 Floating Point Numbers

Floating point numbers follow this regex: `“‘-’?[‘0’-’9’]+‘.’[‘0’-’9’]*”`.
Oscar uses **double** for floating point numbers. A double is a signed 8 byte double-precision floating point data like that in Java. Fractional part can be omitted.

```
double x = 5.4;
double y = -3.4;
double z = 3.;
```

Implicit castings are **not** allowed

```
double error = 3; /* ERROR */
int error2 = 2.4; /* ERROR */
```

Like integers, doubles can be casted down using “AsInt()” method, which truncates away the fractional part of the value. A full

```
double a = 3; /* ERROR */
a == 3; /* ERROR */
AsInt(a) == 3;
```

3.6.1.3 Character

A character is a 1 byte data consisting of ASCII values.

```
char c = ‘c’;
c - ‘f’ == -2; /* true */
```

3.6.1.4 Boolean

Boolean values take 1 byte and can be either `true` or `false.` Other data types cannot be casted as boolean. For example, `none` cannot be used as false

```
bool t = true;
```

3.6.1.5 Unit

Unit type is purely for functions that do not return any value denoted by `unit`

```
def print(s: string) => unit = { }
```

Value/variable of type `unit` cannot be created

3.6.2 Non-Primitive Types

It is worth noting that Oscar treats `list, set and map` as “container primitives”. There are a variety of built-in functions for use with containers, detailed in **section 3.8.**

Keyword	Usage	Example
`string`	String	`string str = “hi”;`
`list`	List backed with HAMT[3] for persistence with performance.	`list<int> intList = list<int>[1, 2, 3];` `list<char> charList = list<char>[];`
`set`	Set of unique values	`set<int> tSet = set<int>[1, 2, 3]`
`map`	Map of key value pairs	`map<int, string> tMap = map<int, string>[0 -> “hi”]`
`actor`	Actor as a first class object	`actor Worker {}`
`message`	Signal between actors and pools	`message(“hi”) |> sender;`
`pool`	Construct used to manage a collection of workers	`pool<Worker> p = spawn pool<Worker>({“worker pool”}, 10);`
`def/func`	Functions	`def f(x: int) => () = return x;`

3.6.2.1 String

[3] http://lampwww.epfl.ch/papers/idealhashtrees.pdf

A string is just a character list. Since implicit type conversion does not exist in Oscar, primitives must be casted by calling `AsString()`.

```
string hi = “hello world”;
hi[0] == ‘h’;
me == “hello world”; /* == on strings compare values */
Size(me) == 11;

Println(“hi :” + 5); /* ERROR */
Println(“hi :” + AsString(5)); /* “hi :5” */
```

3.6.2.2 List

A list is an immutable collection of values of the same type. Lists preserve order, and may hold duplicate elements.

```
list<int> intList = list<int>[]; /* empty list of type int */
list<int> intList = list<int>[1, 2, 3, 4, 5]; /* list literal */
list<double> intList = list<double>[1, 2.0, 3, 4, “5”]; /* ERROR */
```

Since everything without the `mut` keyword is immutable, assignment just returns a new list with assignment applied.

```
list<int> changedList = (initList[3] = -4)
changedList == [1, 2, 3, -4, 5]
initList == [1, 2, 3, 4, 5]
```

If declared with `mut,` lists can be mutated in place:

```
mut list<int> mutableList = list<int>[1, 2, 3, -4, 5]
list<int> alteredList = (mutableList[3] = 5);
mutableList == alteredList == [1, 2, 3, 5, 5]
```

Type list has a few built-in functions. Here are some examples of these methods in action.

```
list<int> exam = list<int>[1, 2, 3, 4, 5];

exam[0] == 1;

listSize(exam) == 5;

list<int> prepended = Prepend(0, exam); /* [0, 1, 2, 3, 4, 5] */

list<int> appended = Append(6, exam); /* [1, 2, 3, 4, 5, 6] */
```

```
 list<int> popFront = PopFront(exam); /* [2, 3, 4, 5] */

 list<int> popBack = PopBack(exam); /* [1, 2, 3, 4] */

 Contains(0, exam); /* true */

list<int> reversed = Reverse(exam) /* [5, 4, 3, 2, 1] */

 Foreach((x:int) => unit = {
       Println(x)  /* prints elements of exam */
 }, exam);

 Filter(exam, (x:int) => int = {
       return x % 2 == 0;
 });
 /* [2, 4] */

 Map((x:int) => int = {
       return x + 2;
 }, exam);
 /* [3, 4, 5, 6, 7] */

 FoldLeft((x:int, y:int) => int = {
       return x + y;
 }, exam);
 /* 15 */
```

3.6.2.3 Set/Map

A set is a collection of distinct elements of the same type. Sets do not preserve order, and hold only one of each item. Examples of using a set are shown below.

```
set<int> set1 = set<int>[1, 2, 3, 4, 5];
set<int> dup = set<int>[1, 2, 3, 4, 5, 1];
set<int> set2 = set<int>[1, 2, 3, 7, 8, 9];

set1 == set2; /* true */

Contains(3, set1) /* true */

Intersection(set1, set2); /* [1, 2, 3] */

Union(set1, set2); /* [1, 2, 3, 4, 5, 7, 8, 9] */

Diff(set1, set2);  /* [4, 5] */
Diff(set2, set1); /* [7, 8, 9] */

set<int> newSet = Put(8, intSet); /* returns a new set */

set == set<int>[1, 2, 3, 4, 5, 8]; /* true */
```

Similarly, maps are collections of key value pairs. Examples of using a map are shown below.

```
map<int, int> test = map<int, int>[];

map<int, double> tMap =
      map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];

map<int, double> tMapCpy =
      map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];

map<int, double> errMap = map<int, double>[0 -> 1];
/* ERROR, type mismatch! */

map<int, double> goodMap = map<int, double>[0 -> 1.2];

Size(tMap) == 3; /* true */

tMap == tMapCpy; /* compares all key-value pairs */

Contains(0, tMap) /* true */

map<int, double> newMap = Put(4, 5.3, tMap);
/* tMap stays, returns a new map */

map<int, double> noZero = remove(0, tMap);
/* returns a new map without key 0 */

Contains(0, noZero) /* false */
```

The syntax for accessing a map is as follows:

```
mapName[key] /* returns value */
```

For example,

```
map<int, double> tMap =
      map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];

double d = tMap[0] /* d == 1.1 */
```

3.6.2.4 Actor

Actors refer to units of concurrency processing defined in the Actor Model.[4] Oscar elevates these into first class constructs.

Actors are special. They can hold functions and *variables,* declared using `mut` keyword. Also, all actors must define `receive` function to handle `message`s. Actors are created with the `spawn` keyword. As explained in Core Concepts section, **section 2.2.2**, values inside actors are private. As such, explicit handling of messages is the only way to affect the state of an actor

```
actor Worker(name: string, initValue: int) {
 /* immutable initializer value */

  mut int x = 10; /* allowed within actors */

  x = 14; /* now x == 14 */

/* all actors must have receive method defined */
  receive = {
    | messageType1() => { ... } /* do something */
    | die() => { Die(); } /* used to kill this actor and
                      all other workers declared in this actor */
  }
}

/* spawn an Actor of type Worker */
actor<Worker> worker = spawn actor<Worker>("name", 3);
```

3.6.2.5 Pool

A `pool` manages a group of actors. When messages are sent into a pool, they are distributed in a round-robin fashion so that many actor(s) can work concurrently. Like actors, pools are created using `spawn` keyword.

```
/* puts ten workers into a pool */
pool<Worker> workerPool = spawn pool<Worker>({"name", 5}, 10);
```

The `pool` constructor takes at least one variable: the number of actors. If an actor requires constructor variables, the last number in a series of formals passed into pool is used as the

[4] https://en.wikipedia.org/wiki/Actor_model

size and the rest are used for actor initialization. Therefore, the syntax for pools is as follows:

```
pool<actorType> poolName =
            spawn pool<actorType>({arg1, arg2 …}, numActors)
```

3.6.2.6 Message

The Message construct is used in inter-actor communications. They can bind values of different types. The send operator `|>` and broadcast operator `|>>` are used to send a message to actors and pools, respectively

```
message helloMessage(prefix: string, suffix: string)
message<helloMessage>("Hello, ", "World!") |> worker;
message<helloMessage>("everyone gets a", " message!") |>>
      workerPool;
/* list of messages can be mass broadcasted */
list<message<helloMessage> >[msg1, msg1, msg1] |>> workerPool
```

`sender` keeps track of the source actor of a message. This can be used to bounce back a message to whomever sent it.

```
message<helloMessage>("hi", "there") |> sender
```

3.6.2.7 Function

There are three syntaxes for defining functions. While each may have it's use, they are all identical in functionality, and thus it is up to the user to use the most appropriate syntax.

The first function syntax is the **top-level syntax**. This uses the `def` keyword. This syntax is often clearest to read, and is encouraged at the top-level of Oscar programs.

```
def identifier(arg: typ1, arg2: typ2 …) => return typ = {
  return retArg;
}
```

For example:

```
def add(a: int, b: int) => int = {
  return a + b;
}
```

The second function syntax is the **lambda syntax**. This syntax is concise, and best used within functions. The lambda syntax is as follows:

```
/* A function f, that takes an int i, and returns the int i+1 */
func f = (i: int) => int = i + 1;
```

The third syntax is the **explicit lambda syntax**. This looks very similar to the lambda syntax, with the addition of types. As it is quite verbose, its use is discouraged. It is as follows:

```
[(int) => int] f = (i: int) => int = i + 1;
```

Since functions are first class objects in Oscar, they can be passed into functions as arguments as well. Because functions are first-order in Oscar, you can simply define the function type to be passed when declaring the function and then either pass a function in by reference as a lambda.

```
/* apply accepts a function f that takes an int and returns an
int, as well as an int input */
def apply(f: [(int) => int], input: int) => int = {
  return f(input);
}

def main() => unit = {
  int a = 41;
  func f = (i: int) => int = i + 1;
  int b = apply(f, a);
  /* b == 42 */
}
```

Just like in the top-level syntax, the body of a function in either of the two lambda syntaxes may consist of multiple statements surrounded by braces, known as a block.

```
def apply(f: [(int) => int], input: int) => int = {
  return f(input);
}

def main() => unit = {
  int b = apply((x: int) => int = {
    Println(x);
    return x + 1;
  }, 5);
```

```
    /* will print 5, and set b == 6 */
  }
```

Functions are then called with the usual syntax of `id(arg1, arg2, …)` as such:

```
  def main() => unit = {
    int x = 39;
    func addTwoNums = (a: int, b: int) => int = a + b;
    int y = addTwoNums(x, 3);
    /* y == 42 */
  }
```

It is possible to declare anonymous lambda functions to pass as arguments, as in the following:

```
  def apply(f: [(int) => int], input: int) => int = {
    return f(input);
  }

  def main() => unit = {
    int a = 52;
    /* pass anonymous decrement-by-10 function */
    int b = apply((x:int) => int = x - 10, a);
    /* b == 42 */
  }
```

No argument functions are declared as such:

```
  def sayHi() => unit = { Println("hi!"); } /* void function */
  sayHi(); /* prints "hi!" */
```

Note that functions must be defined before called. In the same way that you may not use an identifier you have not defined above, you may not use a function you have not defined above. Therefore, it is convention to place your main function at the bottom of your program.

Recursion

Functions can inherently be recursive and call themselves. For instance, to build a list of a certain range you can use the following code:

```
  def foo(x:int) => int = {
    if (x == 0) {
      return 0;
    }
    return x + foo(x-1);
  }

  def main() => unit = {
    int a = foo(3);
    /* a == 3 + 2 + 1 + 0 == 6 */
  }
```

3.7 Program Structure

A typical Oscar program is composed of messages, actors and functions declarations in this *strict* order. This section talks about how execution flows in Oscar.

The structure of an Oscar program is enforced to be a series of messages, actors, and functions, in that order. The main method should be placed at the end of the program.

```
messages
…
actors
…
functions
…
main method
```

This enforced convention keeps all Oscar programs readable and consistent.

3.7.1 Logic and Relations

Oscar has traditional and familiar relational operators for non-container primitive types.

Operator	Usage	Associativity
>	Greater than	None
>=	Greater than or equal to	None
<	Less than	None
<=	Less than or equal to	None

For example:

```
4 > 3 == true;
3.4 < 2.4 == false;
```

Oscar does not allow implicit casting of values. For example, this expression would

```
3.4 < 2;
```

yield an error because value 2 is of type int but 3.4 is of type double. Explicit casting via methods like `AsInt()` and `AsDouble()` must be invoked for logical comparisons across types. These methods are expanded upon in **section 3.8**.

```
3.4 < AsDouble(2);
```

Oscar's logical operations also follow traditional syntax. Though functionality remains largely the same, it is worth pointing out that Oscar does not support implicit conversions of values into boolean values. For instance, type `unit` cannot be evaluated as `false`. Also worth noting that the equality operator `==` extends to container types where every single value is compared.

Operator	Usage	Associativity
`==`	Equal to	None
`!`	Logical Negation	Right
`!=`	Not equal to	None
`&&`	Logical AND	Left
`||`	Logical OR	Left

3.7.2 Control Flow

In Oscar, statements are executed from top to bottom. However this order of execution can be modified using control flow statements.

3.7.2.1 Branching

Conditional branching in Oscar is handled as follows:

```
if (condition) {
  statements block 1
} else {
  statements block 2
}
```

If the first condition is satisfied, first statement is executed, otherwise the statement block in `else` branch is executed. Conditions must be boolean expressions. These can be constructed either by passing a boolean value types as conditions or through relational and logical operations explained previously.

3.7.3 Actor, Pool & Message

Messages are defined as globals so that they can be used by actors. Messages must be declared before any actor declarations. There are a few built-in message and actor types, that will be expanded on in **section 3.10**.

```
message message1(msg: string, payload: double)
message message2(prefix: string, suffix: string)
message message3(prefix: string, suffix: string, extra: string)

actor Actor() {}
...
```

As mentioned before, all actors must have the `receive` handler defined to prevent compilation failures. Inside `receive` a series of pattern matches need to be defined. Values contained in each `message` are destructured and available to its corresponding handler.

```
receive = {
  | message1(msg: string, payload: double) => {
      println(msg + " " + payload.asString());
    }
  | message2(prefix: string, suffix: string) => {
      println(prefix + " " + suffix);
    }
}
```

To prevent an extraneous amount of actors from lingering around, each actor must implement a pattern match for `die` messages. This is a special pattern match, where the actor will immediately cease to exact after the statement block in the pattern match is completely. If there is no code in the body of the pattern match, the actor will still die. Any messages waiting to be process will be discarded. Users must send a `die` message to every spawned actor or pool for cleanup. Without this, program behavior is undefined and could include memory leaks or system hangs.

```
receive {
    | message1(msg: string, payload: double) => {
        println(msg + " " + payload.asString());
      }
    | die() => {
      }
}
```

Once actor types are declared, a `pool` to manage these actors can be declared. These also must be killed by sending the pool a single `die()`

```
pool<Actor> newPool = spawn pool<Actor>({...}, 10);
/* pool of ten Actors */
message<die>() |>> newPool;
/* kill the pool */
```

3.7.4 Executable

Every executable Oscar program needs to define a function named `main()` which takes no arguments and returns `unit`. `main` is a reserved keyword that denotes the first function that gets executed. When multiple Oscar programs are linked together, the compiler will throw an error upon encountering multiple `main()` functions.

```
def main() => unit = { ... }
```

3.8 Built-in Functions

The following section describes the built-in, or library, functions, provided in Oscar. In convention similar to Ocaml, all built-in functions begin with a capitalized letter and follow CamelCase.

3.8.1 The Println Function

Oscar's print function is `Println`. The function is able to print any argument of a primitive type, as well as the container types `list`, `set`, and `map`. Note that `Println` will only accept one argument at a time.

Printing on Primitive Types:

Println Call	Output
`Println("Hello, World!");`	`Hello, World!`
`Println(true);`	`true`

`Println('a');`	`a`
`Println(42);`	`42`

`Println(42.0);`	`42.0000000000`
	`/* No formatting or rounding. */`

Printing on container types is also permitted, and will implicitly call that container's `AsString` function. The `AsString` function is elaborated on in section 3.8.2

Printing on Container Types: (multiple examples)

Println Call	Output
`Println(list<int>[0, 1, 2, 4, 5]);`	`[ 0 1 2 ]`
`Println(list<string>["hey", "hi"]);`	`[ hey hi ]`
`Println(set<int>[1, 2, 3]);`	`[ 1 2 3 ]`

`Println(set<int>[2, 2, 2]);`	`[ 2 ]`
	`/* Only one item is actually in the set. */`

`map<int,int> m = map<int,int>[0->1, 4->5];`
`Println(map);`
	`[ 0 -> 1 4 -> 5 ]`

3.8.2 Type Conversion Functions

Oscar includes several type conversion functions to allow for easy conversion between primitive types. The following table details each conversion function. Each conversion function accepts one argument. For details on the string representation of container types, see section 3.8.1.

Function	Arg type(s)	Return type
`AsInt(arg)`	`double`	`int`
`AsDouble(arg)`	`int`	`double`

`AsString(arg)`	`int`
	`char`
	`bool`
	`double`
	`list`
	`set`
	`map`
		`string`

3.8.3 Container Specific Functions

Oscar includes container specific functions to make complex operations simple. These functions follow functional paradigms, and will return modified copies of the container as opposed to mutating the original container.
For lists and sets, `typ` refers to the type the collection stores, and for Maps, `kTyp` and `vTyp` represent the type of the keys and values. For lists and sets, `item` refers to an item of type `typ`, and for Maps `keyItem` refers to an item of type `kTyp` and `valueItem` refers to an item of type `vTyp`. The general convention for these calling these methods is:

```
Function(arg1, arg2, ... , containerArg)
```

3.8.3.1 ForEach

```
ForEach([(x:typ)=>unit], list) => unit
ForEach([(x:typ)=>unit], set) => unit
ForEach([(k:kTyp, v:vTyp) => unit], map) => unit
```

`ForEach` accepts a function and a container type. `ForEach` will apply the given function to item in the container. In the case of lists and sets, the function provided should take one argument that matches the type of items in the `list` or `set`. In the case of maps, the function should take two arguments that match the types of the key-value pairs in the map. `ForEach` should generally be used with functions that return unit. See 3.8.3.8 for information about the similar function `Map`.

```
/* prints each item in the list */
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  ForEach((x : int) => unit = {
    Println(x);
  }, intList);
}

/* prints each key-value pair in the map */
def main() => unit = {
  map<int, string> myMap
              = map<int, string>[42->”truth”, 41->”nothing”];
  ForEach((x : int, y : string) => unit = {
    Println(x);
    Println(y);
  }, myMap);
}
```

Note that key-value pairs are not ordered, and thus may be printed in a different order than added to the map.

3.8.3.2 FoldLeft

```
FoldLeft((x: a, y: a), a, list) => a'
```

FoldLeft accepts a function, an item, and a list. The specified function must accept two arguments, both of which match the type of the list. The function will be applied to the accumulator and the first item in the list, folding the two into one value. Then the function will be applied to the new value and the next item in the list. This will continue until there is only one item, which is returned.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  int a = FoldLeft((x:int, y:int) => int = {
    return x + y;
  }, 10, intList));
  /* a == 10 + 1 + 2 + 3 + 4 + 5 == 25 */
}
```

3.8.3.3 Reverse

```
Reserve(list) => list
```

Reverse takes a list and returns that list in reverse order.

```
def main() => unit = {
  list<int> l1 = list<int>[1, 2, 3, 4, 5];
  list<int> l2 = Reverse(l1);
  /* l2 == [5, 4, 3, 2, 1] */
}
```

3.8.3.4 Size

```
Size(container) => int
Size(string) => int
```

Size takes a container or string, and returns its size. In the case of list and set, size is defined to be the number of elements. In the case of map, size is defined to be the number of key-value pairs. In the case of string, size is determined to be the number of characters.

```
def main() => unit = {
  map<int, double> tMap =
                map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  int z = Size(tMap));
  /* z == 3 */
```

```
    string s1 = "string1";
    int z = Size(s1));
    /* z == 7 */
  }
```

3.8.3.5 Put

```
Put(item, set) => set
Put(keyItem, valueItem, map) => map
```

Put adds an item to a set, or adds adds a key-value pair to a map. For sets, if the item being added is already in the set, then the returned set will be identical to the one given. For maps, if the key being added already exists in the map, its value will be overridden with the given value.

```
def main() => unit = {
  map<int, double> tMap =
               map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  map1<int, double> map1 = Put(2, 1.3, tMap);
  /* map1 now contains the k->v pair 2 -> 1.3 */
  map2<int, double> map2 = Put(0, 4.2, map1);
  /* map2 overwrites and now contains the k->v pair 0 -> 4.2 */
}

def main() => unit = {
  set<int> tSet = set<int>[1, 2, 3, 4, 5];
  set<int> set1 = Put(7, tSet);
  /* set1 contains [1, 2, 3, 4, 5, 7] */
  set<int> set2 = Put(4, set1);
  /* set2 contains [1, 2, 3, 4, 5, 7], no change */
}
```

3.8.3.6 Filter

```
Filter([(x:typ)=>bool], list) => list
Filter([(x:typ)=>bool], set) => set
Filter([(k:kTyp, v:vTyp)=>boo]l, map) => map
```

Filter takes a function that returns a boolean value and a collection. It will apply the given function to each item or key-value pair in the given collection, and add it to the returned collection if the function returns `true`.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = Filter((x:int) => bool = (x % 2) == 0, intList)
  /* l is [2, 4] */
}
```

3.8.3.7 Contains

```
     Contains(item, list) => bool
     Contains(item, set) => bool
     Contains(keyItem, map) => bool
```

Contains is a simple method that returns true if the given item exists in the collection. In the case of a map, the function returns true if the key given exists in the map.

```
def main() => unit = {
  set<int> intSet = set<int>[1, 2, 3, 4, 5];
  Bool b = Contains(3, intSet);
  /* b == true */
  bool c = Contains(0, intSet);
  /* c == false */
}

def main() => unit = {
  map<int, double> tMap =
                map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  bool b = Contains(0, tMap);
  /* b == true */
  bool c = Contains(1, tMap);
  /* c == false */
}
```

3.8.3.8 Map

```
     Map([(x: a)=> a’], list<a>) => list<a’>
     Map([(x: a)=> a’], set<a>) => set<a’>
     Map([(k: a, v: b)= b’, map<a, b>) => map<a, b’>
```

Map accepts a container and a lambda function, and returns a container consisting of the result of calling that function on each element in the container. For example:

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = Map((x:int) => int = {
    return x + 2;
  }, intList);
  /* l is [3, 4, 5, 6, 7] */
}
```

When calling Map on the map container, a lambda function that accepts two arguments is necessary. This function must return a type matching the type of the values. Therefore, it is only possible to modify the values of a map with the Map function, and not the keys.

```
def main() => unit = {
  map<int, int> intMap = map<int, int>[0 -> 1, 4 -> 5, 3 -> 5];
  map<int, int> m = Map((k:int, v:int) => int = {
    Return x + y;
  }, intMap);
  /* m is [0 -> 1 4 -> 9 3 -> 8] */
}
```

3.8.3.9 Reduce

```
Reduce([(x: a)=> a’], list<a>) => item: a’
```

This function is very similar to `FoldLeft`, except it does not take an accumulator, or starting value.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  int a = Reduce((x:int, y:int) => int = {
    return x + y;
  }, intList));
  /* a == 1 + 2 + 3 + 4 + 5 == 15 */
}
```

3.8.3.10 MergeFront

```
MergeFront(list<a>, list<a>) => list<a>
```

This function accepts two lists, and returns all the items in the first list followed by the items in the second list. For example:

```
def main() => unit = {
  list<int> lst1 = list<int>[1, 2, 3, 4, 5];
  list<int> lst2 = list<int>[9, 8, 7];
  list <int> lstBoth = MergeFront(lst1, lst2);
  /*lstBoth is [9, 8, 7, 1, 2, 3, 4, 5] */
}
```

3.8.3.11 MergeBack

```
MergeBack(list<a>, list<a>) => list<a>
```

This function accepts two lists, and returns all the items in the second list followed by the items in the first list. For example:

```
def main() => unit = {
  list<int> lst1 = list<int>[1, 2, 3, 4, 5];
  list<int> lst2 = list<int>[9, 8, 7];
  list <int> lstBoth = MergeBack(lst1, lst2);
  /*lstBoth is [1, 2, 3, 4, 5, 9, 8, 7] */
}
```

3.8.3.12 PopFront

```
PopFront(list<a>) => list<a>
```

Popfront simply returns a list without the first element. As lists are backed by the aforementioned HAMTs, this is an O(1) operation.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = PopFront(intList);
  /* l is [2, 3, 4, 5] */
}
```

3.8.3.13 PopBack

```
PopBack(list<a>) => list<a>
```

Popfront simply returns a list without the last element. As lists are backed by the aforementioned HAMTs, this is an O(1) operation.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = PopBack(intList);
  /* l is [1, 2, 3, 4] */
}
```

3.8.3.14 Prepend

```
Prepend(item: a, list<a>) => list<a>
```

Prepend simply adds an item to the front of the list.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = Prepend(0, intList);
  /* l is [0, 1, 2, 3, 4] */
}
```

3.8.3.15 Append

```
Append(item: a, list<a>) => list<a>
```

Append simply adds an item to the back of the list.

```
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  list<int> l = Append(6, intList);
  /* l is [1, 2, 3, 4, 5, 6] */
}
```

3.8.3.16 Union

```
Union(set1<a>, set2<a>) => set<a>
```

Union returns the union of the two sets. That is, the items that are in either set.

```
def main() => unit = {
  set<int> intSet1 = set<int>[0, 1, 2, 3];
  set<int> intSet2 = set<int>[0, 4, 5];
  set<int> u = Union(intSet1, intSet2);
  /* u is [0, 1, 2, 3, 4, 5] */
}
```

3.8.3.17 Diff

```
Diff(set1<a>, set2<a>) => set<a>
```

Diff returns the the items that are in the first set but not in the second set. Note that the order of arguments matters. A set diffed with itself returns the empty set.

```
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  set<int> intSet2 = set<int>[0, 3, 4, 6];
  set<int> d1 = Diff(intSet1, intSet2);
  /* d1 is [1, 2, 5] */
  set<int> d2 = Diff(intSet2, intSet1);
  /* d2 is [0, 6] */
  set<int> d3 = Diff(intSet1, intSet1);
  /* d3 is [] */
}
```

3.8.3.18 Intersection

```
     Intersection(set1<a>, set2<a>) => set<a>
```

Intersection returns the intersection of the two sets. That is, the items that are common to both sets.

```
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  set<int> intSet2 = set<int>[0, 3, 4, 6];
  set<int> s1 = Intersection(intSet1, intSet2);
  /* s1 is [3, 4] */
}
```

3.8.3.19 Subset

```
     Subset(set1<a>, set2<a>) => bool
```

Subset returns true if the first set is a subset of the second set. That is, all of the elements in the first set are present in the second set.

```
def main() => unit = {
  set<int> small = set<int>[3, 4];
  set<int> big = set<int>[1, 2, 3, 4, 5];
  bool b = Intersection(small, big);
  /* b == true */
}
```

3.9 Scoping and Immutability

Oscar is a statically scoped language. This means that scope is determined by textual structure as opposed to runtime behaviour. Scopes are effectively delineated with pairs of curly braces. In the below example, we can see that b is defined in the inner scope, and then the variable goes out of scope at the end of its closure. This allows for b to be redefined again later.

```
def main() => unit = {
  int a = 5;
  {
    Println(a); /* 5 is printed, a is available in this scope */
    int b = 6;  /* b is defined */
    Println(b); /* 6 is printed */
  }
  Println(a);  /* 5 is printed */
  int b = 4;   /* a new b is defined */
  Println(b);  /* 4 is printed */
}
```

Scopes are generally open, meaning that they inherit from their surrounding scopes. Because scopes are open, it means that it is not possible to “hide” variables by declaring variables of the name name in a tighter scope.

```
def main() => unit = {
  int b = 4;
  {
    int b = 6;  /* b is already defined! Error */
    Println(b);
  }
  Println(b);
}
```

Variables and collections are by default immutable in Oscar. Mutable variables are only allowed in Actors, and are defined with the mut keyword. These variables are allowed to be reassigned and modified at will.

```
message testNow()

actor Tester() {
  mut int i = 0;
  receive = {
    | testNow() => {
        i = i + 1;
    }
  }
}
```

3.10 Standard Library

Oscar includes a (currently very small) standard library, in which helpful messages, actors, and functions are placed. For instance, there are currently two messages:

```
message die()
message print(msg: string)
```

There is also a built in `Printer` Actor, that will print any `print` message sent to it to stdout. This could be expanded in the future to allow for simple writing to files or reading from streams.

4. Project Plan

4.1 Planning Process

Oscar was developed in a flexible, ad-hoc fashion throughout the semester. Initially we designated Wednesday nights as our primary working time - this coincided with our weekly meetings with David Watkins, during which we discussed any recent coding problems or challenges that we had encountered - and as the semester progressed we included Saturdays in our designated group working time. The majority of time was initially spent working on the language specifications, designing and programming the scanner, parser, ast, analyzer, and codegen, and lastly incorporating testing too. Using GitHub, we were able to effectively work individually on separate branches when necessary and subsequently merge our work with the master branch upon receiving group approval, or after making group-requested changes/corrections if necessary.

4.2 Specification

At the beginning of the semester we had originally intended for *Oscar* to be an Actor-Oriented programming language, the primary use of which would be for programs requiring parallel computation. Through message-passing from functions and/or actor objects to actors, multiprocessing would be handled by bespoke actors working on separate tasks or subtasks simultaneously. As actors are each able to encapsulate their own mutable data, threading would be relatively simple and handled under the hood.

4.3 Development Process

Checkpoints for our code completion were mapped out by the deliverable deadlines set at the beginning of the course. As anticipated, we did have to make multiple changes to our initial Language Reference Manual specifications in order to meet some of these deadlines and to keep our workload from becoming insurmountable. After completing our Scanner, Parser, and Abstract Syntax Tree, we turned towards basic code generation to meet the Hello World deadline. After accomplishing that milestone, we returned to semantic analysis to provide type checking of our programs and built a semantically checked Abstract Syntax Tree, or SAST. Finally, we wrote the necessary C++ bindings for our language, and linked them in with our code generation. This was largely because we realized too late that immutable structures library written in C++ relied heavily on template metaprogramming but since these templates are finalized at compile time, binding with LLVM meant that we either scrap these entirely and write everything in raw C, which would have taken just as long not to mention actor implementation that we had not started or just transpiling to C++ and produce LLVM IR through Clang. In end end we decided that the latter approach would be far more productive and make our language more useful.

4.4 Testing Process

Initially we created some scanner test files consisting of sample *Oscar* code along with files for their expected scans to ensure that our parsing process was working as expected, and once we had a relatively concrete blueprint for the syntax and functionalities of *Oscar*, we added compiler unit tests along with files for their expected outputs for each individual aspect of our language. Lastly, we added compilation error tests along with files for their expected error messages to ensure that our analyzer handled failures as we expected.

4.5 Team Responsibilities

At the beginning of the semester each team member was assigned a specific responsibility: Manager, Language Guru, System Architect(s) and Tester. The following table lists each team member's areas of contribution within the our completed project.

Team Member	Responsibilities
Ethan Adams	Parser, Codegen, Compiler Top-level, Final Report
Howon Byun	Analyzer, Transpiler, C++ bindings
Jibben Hillen	Parser, Codegen, Transpiler, Optimizer
Vladislav Scherbich	C++ bindings, Final Report
Anthony Holley	Testing Suite, Final Report, Presentation

4.6 Software Development Environment

- **Git:** version control system
- **Ocaml 4.03.0:** used for scanning, parsing and semantic checking
- **Corebuild**
- **Makefile:** used for compiling, linking and testing
- **C++ 14**
- **Clang++ 3.7(or above):** compiling generated C++ files into executables.
- **Bash Shell Scripting:** used for testing automation

4.7 Programming Style Guide

The Oscar team adopted agreed upon code standards throughout our development. Largely, these standards follow general Ocaml practice. Indentation of 2 spaces was adopted in Ocaml files. Code should be clear and concise, and largely self-explanatory. Comments were included at the start of complex blocks and functions. These standards were enforced in the code review stage on each branch prior to merging.

4.8 Project Timeline

Date	Milestone
Sep 28th	Project Proposal complete

Oct 26th	Language Reference Manual complete
Nov 21st	Hello World test passes, Test suite developed
Dec 15th	Compiler complete
Dec 20th	Final Report complete

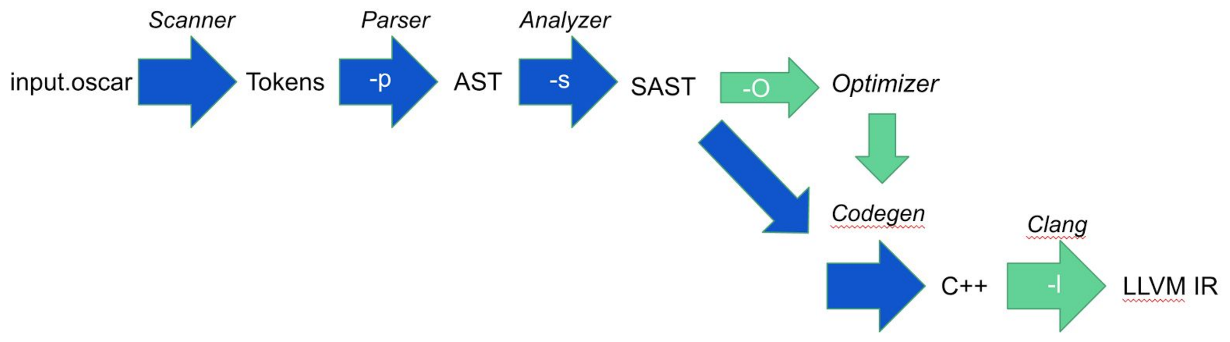
4.9 Project Log

Date	Milestone
Sep 28th	Project Proposal complete
Oct 26th	Language Reference Manual complete
Nov 5th	Scanner/Parser complete first time
Nov 18th	Hello World test passes, Test suite developed
Dec 3rd	Semantic Analysis complete
Dec 10th	Codegen functional
Dec 13th	Test Suite Refactored
Dec 18th	Optimizer complete; Compiler top-level complete; C++ bindings complete
Dec 20th	Codegen complete
Dec 20th	Final Report complete

5 Architecture

5.1 The Compiler

The Oscar compiler consists of a scanner, parser, prettyprinter, semantic analyzer, transpiler, and optimizer. The interaction between this components is shown in the diagram below:



5.1.1 Scanner (Jibben, Howon)

The scanner takes an Oscar program and tokenizes it into the defined tokens. This is largely simple, the only complex section being ensuring correct regexes for identifiers, strings, characters, and allowing for escaped characters.

5.1.2 Parser (Jibben, Ethan, Howon, Vlad)

The Parser takes the tokenized program and runs it through a Context Free Grammar (CFG), and builds up an Abstract Syntax Tree (AST) to represent that program. Associativity is defined for most if not all operators to avoid ambiguity, and special attention is paid to developing a syntax that precludes shift/reduce errors.

5.1.3 Pretty Printer (Jibben, Anthony, Howon)

The Pretty Printer takes a AST and prints it out in the format of the originally written code. This is a helpful tool for determining if a program is scanned and parsed correctly. If a program is parsed correctly, then the input program will be exactly the same as the result of the pretty printer (minus some formatting).

5.1.4 Semantic Analyzer (Howon, Jibben)

The Semantic Analyzer performs a Postorder traversal of the AST and transforms it into a Semantically checked Abstract Syntax Tree, or SAST. The SAST has types associated with each expression. For example, the result of the addition of two ints has the type `int` associated with it. The analyzer is also helpful in confirming that types match correctly; that functions are passed the correct types, operators are provided matching types, and so on. This is accomplished by recursing on provided arguments and ensuring that types match at each level. If types do not match, then this is a stage which an error will be generated.

5.1.5 Code Generator (Howon, Jibben)

Code Generation is done similarly to Semantic Analysis. The SAST is traversed, and at each step, the given oscar construct is matched with its appropriate C++ construct. This allows for efficient and simple generation of C++ code. Additional C++ codes were injected at this step such as destructors for messages and actors and a global Monitor actor that keeps track of all actors in use.

5.1.6 Optimizer (Jibben)

Every Oscar program is run through a two-pass optimizer at compilation time. The following factors are optimized:

- All immutable primitive value declarations are removed and their later references are replaced with their declared values
- All binary operators and unary operators with expressions that have been optimized to literals are evaluated into a single literal
- All unused code blocks are removed
- if/else conditionals are evaluated (if the predicate can be optimized to a literal) and all unused code is discarded
- All unused function declarations and variable declarations of all forms are removed

For the purpose of clarity in this section, oscar programs and their resulting optimized C++ code will be shown.

Oscar’s nature as a largely immutable language lends well to optimization. Immutability makes expression folding relatively simple. Given constants are folded together during compilation to reduce steps in resulting C++ and assembly. This is clear in the example below:

```
optFold.oscar:
      def main() => unit = {
        Println(3 + 2);
      }

optFold.cpp:
      int main ()
      {
            Println(5); /* folded! */
            return 0;
      }
```

By making use of referential transparency, identifiers can often be replaced by the literal, that is, the value they they have been assigned to. This removes unnecessary load and store operations.

```
optReplace.oscar:
      def main() => unit = {
        int x = 7;
        Println(x);
      }
optReplace.cpp:
      #include <oscar/immut.hpp>
      int main ()
      {
            Println(7); /* x is replaced by its value */
                          /* Println is implemented in immut.hpp */
            return 0;
      }
```

By replacing identifiers and then folding, it is possible to transform complex expressions into a single constant.

Oscar compiler also removes “dead code”: unused variables, functions, and branches that are guaranteed not to be taken. A example of all of these optimizations together is shown below.

```
optAll.oscar:
      /* unused function */
      def foo() => unit = {
            return;
      }

      def main() => unit = {
        int x = 5;
        int y = 6;
        int z = 7; /* unused variable */
        y;        /* useless statement */
        if (false) { /* unnecessary branch */
            Println(0);
        } else {
          Println(x + y + 3 + 2);
        }
      }

optAll.cpp:
      #include <oscar/immut.hpp>
      int main ()
      {
            Println(16);
            return 0;
      }
```

5.2 The C++ Libraries (Howon, Vlad)

Much of Oscar’s rich functionality comes from its strong libraries implemented in C++. By utilizing templates extensively, builtin functions in Oscar are implemented in a consistent fashion. This section gives a brief overview of each of the library files and its purpose.

5.2.1 Immut.hpp, List.hpp, Map.hpp, Set.hpp, Collection.hpp

These files define the immutable collections that back Oscar. Each file defines the collection respectively. Immut.hpp defines all of the built-in functions that are described in **section 3.8.**

Lists are implemented as a singly linked list of nodes that are wrapped in `std::shared_pointer<T>` to allow for efficient shared references to each node as well as automatic garbage collection.

Sets and Maps are implemented as Red-Black Trees for logarithmic time modification operations. All necessary operators are overloaded to allow nested container types.

All of the builtin functions described above are implemented in this header as well. They also make use of templates to reduce the amount of code we write while making this generic and implementation clean.

5.2.2 Actor.hpp

This file defines the Actor object that backs Actors in Oscar. Actors implement something similar to the producer-consumer paradigm, where the current actor is a consumer of other actors messages (producers). Internally, messages are stored in a queue as they arrive from other actors. This queue is drained as messages arrive and are consumed. Synchronization is achieved via use of threads, condition variables and signals.

5.2.3 Message.hpp

Messages is a light-weight class that consists of name and an optional sender field (an actor pointer). The receiving actor can utilize the sender field to return an answer to the message sender after processing.

6 Testing

The test suite for *Oscar* has three main sections: scanner tests, compiler tests and compiler-error tests. The tests in the scanner section consist of supplementary *Oscar* code and collectively ensure that every feature within the language scans and parses correctly.

The tests in the compiler section consist of unit test cases, each of which checks a specific feature of *Oscar* by comparison of the output from a print statement within the test code and a file containing the expected output. In the same way, tests in the compiler-error section check specific error messages for different cases of invalid user input.

6.1 Testing Phases

6.1.1 Unit Testing

Our test suite contains unit tests for every individual aspect of *Oscar*. The benefit of having a large number of narrow-scope tests like this is that whenever a test fails, the aspect of the language which caused the failure is immediately clear. Upon failing, tests whose outputs are dependent on multiple features of the language have ambiguous cause for failure. While sometimes having more than one dependency (through necessity), our compiler-error tests are also as narrow-scope as possible for this reason.

6.1.2 Integration Testing

Our test suite contains integration tests in both the scanner and compiler sections. These tests are larger in scope and their outputs have many dependencies. Once all corresponding unit tests have passed, the integration tests confirm that the individual features of *Oscar* function correctly in unison, as well as individually.

6.2 Automation

Automated testing was implemented through our `test.sh` file. Run the test script from the `test` directory with the following command:

```
$ ./test.sh
```

The script will run through all tests, and print the name of the test and its success or failure. Succeeding tests are printed in green, and failing tests are printed in red. A summary of the tests is provided at the end of the script.

The Test Suite works by compiling the given `.oscar` file and generating an executable. The executable is run, and the output is “diffed” with the expected output in the `.oscar.out` file. The Test Suite also tests for the limitations of our language. In the `test/oscar/compiler/errors` directory, there are tests that are designed to fail. The tests are run through the compiler, and the error produced is “diffed” against the expected error, in the `.oscar.out` file.

A verbose breakdown of the testing results is made available in the file `logfile.log`. Each test may result in messages generated from the Oscar compiler or from the Clang compiler. As described in **section 2.1**, Oscar compiler messages may include failed scanning, parsing, or analysis. Clang messages may include warnings or errors. The Oscar transpiler is designed such that no Clang messages will be generated, so in the case that there are, it is likely that the oscar code provided is at fault.

In the file `logfile.log`, a passed test will look like the following:

```
==================================
Testing Compiler: AND
Oscar Compiler Messages:
Clang Messages:

AND passed
```

Note that as the test suite also tests error cases, if a test is intended to generate a specific error, and does generate the desired error, then the test is considered “passing”.

A test may fail in several ways. The oscar compiler may generate an error. In this case, no C++ file or executable is generated, and the test will have no output. The contents of the failing test is also displayed. In the file `logfile.log`, this will look like the following:

```
==================================
Testing Compiler: arithBinOps
Oscar Compiler Messages:
Error: operand type mismatch: int + double

arithBinOps failed $
def main() => unit = {
  Println(1 + 2.9);
  Println(1.0 + 2.0);
}

Generated Output:
```

A test may also fail by having different output than the expected. In this case, the generated output is displayed along with the failing test. In the file `logfile.log`, this will look like the following:

```
==================================
Testing Compiler: arithBinOps
Oscar Compiler Messages:
Clang Messages:

arithBinOps failed $
def main() => unit = {
  Println(1 + 7);
}
```

```
Generated Output:
17
/* unexpected output */
```

6.3 Test Suite

The Test Suite covers the following features of our language:

- ❏ Binary operations
    - ❏ Arithmetic operations, boolean operations, and bitwise operations.
- ❏ Unary operations
    - ❏ Integer negation and the logical Not.
- ❏ Primitive types
    - ❏ Assignment and printing
    - ❏ String operations
- ❏ Control Flow
    - ❏ If, else
    - ❏ Nested if else clauses
- ❏ Scope
    - ❏ Allowing for identifiers of the same name in different scopes
- ❏ Functions
    - ❏ Passing arguments to a function
    - ❏ Returning from a function
- ❏ Lambdas
    - ❏ Assigning lambdas to identifiers
    - ❏ Passing lambdas to other functions
- ❏ Container types: `map`, `set`, and `list`
    - ❏ Initialization of each respective type
    - ❏ Comparison between types with ==
    - ❏ Builtin Functions used to manipulate Container Types

- ❏ Builtin Functions
    - ❏ Used with container types
    - ❏ Type conversions between primitives
    - ❏ Println
    - ❏ AsString
- ❏ Actor
    - ❏ Pingpong
    - ❏ Mutability tests

The errors tested for are as follows:

- ❏ Type mismatches
    - ❏ Assignment of a literal to an identifier of a different type
    - ❏ Use of incorrect types with an operator
    - ❏ Passing parameters of the wrong type to a function
    - ❏ Constructing actors with bad arguments
    - ❏ Constructing messages with bad arguments
    - ❏ Returning the wrong type in a function
        - ❏ Note: not returning is identical to returning `unit`
- ❏ Identifiers
    - ❏ Use of an undeclared identifier
    - ❏ Declaring an identifier twice in the same scope
    - ❏ No `main` function
- ❏ Immutability/mutability
    - ❏ Immutability disallowed outside actors
    - ❏ Mutability allowed inside actors
- ❏ Scope
    - ❏ Variables accessible only within appropriate scope
    - ❏ Functions unable to access other function’s variables
    - ❏ Actors unable to access other actor’s variables

7 Team Experience

7.1 Howon

Even though I hated every second of sleep I lost trying to make our language as functional(pun intended) as possible during the very last weeks of this semester, I can confidently say PLT has been the most rewarding and enriching class I have taken so far. This course far exceeded my expectation in terms of the breadth of material actually relevant in real world, both in terms of technical skills I gained and working with teammates.

Advice:
Don't take PLT and OS at the same time. As professor said, great languages are produced from dictatorship and maintaining focus rather than trying to please everyone. Aiming for a moving target sucks and is counterproductive. Be honest with your teammates about your strengths and weaknesses so that you don't become a liability. Don't be defensive or get stuck thinking that your code is good before having it thoroughly reviewed. Be willing to take on more mundane tasks as they are all integral to creating a usable language.

7.2 *Jibben*

The project of writing your own language from scratch really seemed daunting at the start of the semester, and for good reason. This project proved to be very challenging and some compromises had to be made along the way. While at every stage it felt as if it would get easier going forward, in reality it only got harder. In accordance with this, we attempted to start early, but definitely picked up the pace as the water started to boil towards the end of the semester. Looking back it feels as if we could have been far more productive and speedy with the first parts of the project (scanner, parser, ast), but that is just the nature of learning and software development in general.

This class has been great overall, and I have learned a ton about programming language theory, compilers, and especially functional programming. During the last crunch week, I dreamt in OCaml - no lie. Professor Edwards is right when he says that OCaml is the best tool for writing a compiler. I cannot imagine trying to do all of this with another language, even one that I had better knowledge of going into the semester.

Advice:

Everyone says this, but, really, ***START EARLY***. Staying up all night for all of reading week to finish this project was not a fun experience and made studying incredibly difficult. Also, be honest about your abilities and time commitments - pretending to be able to do more than you can for your team is only going to hurt everyone in the long run. And if you say you're going to get something done, then get it done. A half-completed module does not do much good for the rest of the team. Lastly, try to write code (modularly) such that other people can also write on the same module at the same time. If you have to do a big overhaul or refactor, just bite the bullet and do it as fast as possible so other people can get back to working on the file/project.

7.3 *Ethan*

At the start of the course, writing our own language seemed unimaginable, if not farfetched. We started with very little knowledge of how compilers worked and almost little experience in Ocaml. However, after a brief planning period, we were able to quickly decide upon a language and get to work. In retrospect, I would say we somehow spent a large amount of our time in the Parsing stage, despite later having to come back and change our Parser more than once. This was partly due to a lack of understanding, a lack of foresight, and a lack of code quality on my part. I think it would have been best to take this time to clearly lay out our language and its features, such that the Parser would not have had to change.

I had some difficulty working in Ocaml, although I got used to the language as the project continued. Some of the functional Ocaml code I wrote was difficult to modularize, which led to a majority of Codegen being refactored and restructured entirely.

I'd say I've learned much from this course as a whole. I have a much better understanding of Compilers and functional programming. I've also learned the importance of teamwork, and good communication practices as well as good code practices. You need to talk to and complement your teammates, as well as have your code complement theirs.

Advice:

Ensure you're dividing work well. Make sure your code does not affect another's (if possible), and treat the project iteratively. Write your code to be easily modularized, because things will need to change often. Follow through when you say you'll do something, and if you can't, then raise that concern as soon as possible.

7.4 *Vlad*

I feel like this class was one of the most useful CS classes I've taken in my career. It's also been one of the hardest. But the benefits of taking it outweigh the negatives, in my view. Although not an expert in the field of compilers, I have a much better understanding of how languages are designed and written, and how the translation/compilation process goes through stages. This class made me appreciate how much thought and planning goes into writing a language (unless it's Javascript). Aside from all that, the group project should teach (or reinforce) good engineering principles, which is extremely useful in the industry.

Advice:
When deciding on a language to implement, think of an interesting problem you'd like to solve. It's a good idea for every member of the group to come up with a few options, and describe them in the form of a mini-proposal (a paragraph or two). This will help you solidify ideas in your mind, as well as present them to your teammates in a concise form. Then cast a vote! This is how we ended up picking a language to implement. And then start implementing the language early! I cannot stress this enough: start early. You'll need the extra time to revise your language spec for the hundredth time. Make sure that everyone's in the loop about specifications and understands what your language looks like. Follow good software engineering practice, i.e. use version control, do code reviews, write tests, etc. Don't push changes that break you code. Just don't do it. It slows everybody down and is a major cause of frustration. And lastly, have fun! You're getting to write a language, and not many can boast about having done that :)

7.5 *Anthony*

I can't express enough gratitude to the other members of my team for all of the occasions on which they have helped me to continually learn more throughout the semester. Due to class scheduling conflicts (senior year computer engineering major), I am taking this course prior to taking 3157 (Advanced Programming) and the learning curve has been steep to say the least. Having never worked on a group-based programming task before, I have definitely gained my most invaluable lessons in this class through all of the team-oriented aspects of the project

(including GitHub). Additionally, learning more about and working through the testing stages for a project of this scope has opened my eyes to how indispensable this process is to the success and overall efficiency of large-scale
coding endeavors like this language.

Advice:
Start writing your testing suite in unison with the rest of the code (you will thank yourself down the line); know GitHub.

8 Code Listing

8.1 Oscar Files

8.1.1 scanner.mll

```
{
  open Parser
  exception Scan_error of string

  let make_char c =
    let ec = Char.escaped c in
    let s = Scanf.unescaped (match String.length ec with
        1 -> "\\" ^ ec
      | _ -> ec) in
    String.get s 0

  let fail c = raise (Scan_error (Char.escaped c))
}

let digit = ['0' - '9']
let double = ('-'?)((digit+'.'digit*) | ('.'digit+))
let simp_chr = [' '-'!' '#'-'&' '('-'[' ']'-'~']
let esc_chr = ['t' 'n' 'r' '\'' '\"' '\\']
let all_chr = simp_chr | esc_chr

rule token = parse
    [' ' '\t' '\r' '\n']  { token lexbuf }
  | "/*"                  { comment lexbuf }

  (* braces/parens/brackets *)
  | '('                   { LPAREN }
  | ')'                   { RPAREN }
  | '{'                   { LBRACE }
  | '}'                   { RBRACE }
  | '['                   { LBRACKET }
  | ']'                  { RBRACKET }
  | '<'                  { LANGLE }
  | '>'                  { RANGLE }

  | '='                  { ASSIGN }
  | "->"                 { ARROW }

  (* math *)
  | '+'                  { ARITH_PLUS }
  | '-'                  { ARITH_MINUS }
  | '*'                  { ARITH_TIMES }
  | '/'                  { ARITH_DIVIDE }
  | '%'                  { ARITH_MOD }

  (* comma, semi *)
  | ','                  { PUNC_COMMA }
  | ';'                  { PUNC_SEMI }

  (* logic *)
  | "=="                 { LOGIC_EQ }
  | "!="                 { LOGIC_NEQ }
  | "<="                 { LOGIC_LEQ }
  | ">="                 { LOGIC_GEQ }
  | "&&"                 { LOGIC_AND }
  | "||"                 { LOGIC_OR }
  | "true"               { LOGIC_TRUE }
  | "false"              { LOGIC_FALSE }
  | '!'                  { LOGIC_NOT }

  (* bit math *)
  | '&'                  { BITWISE_AND }
  | '|'                  { BITWISE_OR }
  | '^'                  { BITWISE_XOR }
  | ">>"                 { BITWISE_RIGHT }
  | "<<"                 { BITWISE_LEFT }
```

```
(* functions *)
| ':'                   { FUNC_ARG_TYPE}
| "=>"                  { FUNC_RET_TYPE }
| "return"              { RETURN }

(* flow control *)
| "if"                  { FLOW_IF }
| "else"                { FLOW_ELSE }

(* actors *)
| "spawn"               { ACT_SPAWN }
| "receive"             { ACT_RECEIVE }
| "|>"                  { ACT_SEND }
| "|>>"                 { ACT_BROADCAST }

(* mutability for actors *)
| "mut"                 { MUTABLE }

(* primitive types *)
| "int"                 { TYPE_INT }
| "double"              { TYPE_DOUBLE }
| "char"                  { TYPE_CHAR }
| "bool"                { TYPE_BOOL }
| "unit"                { TYPE_UNIT }

(* function declaration *)
| "def"                 { FUNC_DECL }

(* non-primitive types *)
| "string"              { TYPE_STR }
| "list"                { TYPE_LIST }
| "set"                 { TYPE_SET }
| "map"                 { TYPE_MAP }
| "message"             { TYPE_MESSAGE }
| "actor"               { TYPE_ACTOR }
| "pool"                { TYPE_POOL }
```

```
  | "func"                { TYPE_FUNC }

  (* literals *)
  | digit+ as lxm { INT_LIT(int_of_string lxm) }
  | double as lxm { DOUBLE_LIT(float_of_string lxm)}
  | '\"' ((simp_chr | '\\' esc_chr)* as lxm) '\"' { STRING_LIT(lxm) }
  | '\'' (simp_chr as lxm) '\'' { CHAR_LIT(lxm) }
  | "'\\" (esc_chr as ec) "'" { CHAR_LIT(make_char ec) }
  | "true" | "false" as lxm { BOOL_LIT(bool_of_string lxm) }
  (* identifiers *)
  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_' '-']* as lxm { ID(lxm) }
  | eof { EOF }
  | _ as bad { fail bad }

and comment = parse
  | "*/" { token lexbuf }
  | _    { comment lexbuf }
```

8.1.2 Ast.ml

```
type bin_op =
    Add | Sub | Mult | Div
  | Mod | Equal | Neq | Less
  | Leq | Greater | Geq | And
  | Or | Bit_And | Bit_Or | Bit_Xor
  | Bit_RShift | Bit_LShift | Assign

type u_op = Not | Neg

type types = Int_t | Bool_t | Double_t | Char_t | Unit_t | String_t
  | Func_t     of types list * types
  | List_t     of types
  | Set_t      of types
  | Map_t      of types * types
  | Actor_t    of string
  | Pool_t     of string
  | Message_t  of string

and expr =
    Int_Lit      of int
  | Double_Lit   of float
  | Char_Lit     of char
  | String_Lit   of string
  | Bool_Lit     of bool
  | Unit_Lit     of unit
  | Id           of string
  | Access       of expr * expr
  | Func_Lit     of func
  | List_Lit     of types * expr list
  | Set_Lit      of types * expr list
  | Map_Lit      of types * types * (expr * expr) list
  | Actor_Lit    of string * (expr list)
  | Pool_Lit     of string * (expr list) * expr
  | Message_Lit  of string * (expr list)
  | Binop        of expr * bin_op * expr
  | Uop          of u_op * expr
  | FuncCall     of string * expr list
  | Noexpr

and stmt =
    Block       of stmt list
  | Expr        of expr
  | Return      of expr
  | Vdecl       of val_decl
  | Mutdecl     of mvar_decl
  | If          of expr * stmt * stmt
  | Actor_send  of expr * expr
  | Pool_send   of expr * expr

and formal = string * types

and message = {
    m_name: string;
    m_formals: formal list;
}

and val_decl = {
    v_name : string;
    v_type : types;
    v_init : expr;
}

and mvar_decl = {
    mv_name : string;
    mv_type : types;
    mv_init : expr;
}

and func = {
  f_formals  : formal list;
  f_return_t : types;
  f_body     : stmt;
}

and pattern = {
  p_mid       : string;
  p_mformals  : formal list;
  p_body      : stmt;
}

and actor = {
  a_name      : string;
  a_formals   : formal list;
  a_body      : stmt;
  a_receive   : pattern list;
}

type program = message list * actor list * val_decl list

(* PRETTY PRINTER *)

let str_binop = function
    Add         -> "+"
  | Sub         -> "-"
  | Mult        -> "*"
  | Div         -> "/"
  | Mod         -> "%"
  | Equal       -> "=="
  | Neq         -> "!="
  | Less        -> "<"
  | Leq         -> "<="
  | Greater     -> ">"
  | Geq         -> ">="
  | And         -> "&&"
  | Or          -> "||"
  | Bit_RShift  -> ">>"
  | Bit_LShift  -> "<<"
  | Bit_And     -> "&"
  | Bit_Or      -> "|"
  | Bit_Xor     -> "^"
  | Assign      -> "="

let str_uop = function
    Not  -> "!"
  | Neg  -> "-"

let rec str_types = function
    Int_t                -> "int"
  | Bool_t               -> "bool"
  | Double_t             -> "double"
  | Char_t               -> "char"
  | Unit_t               -> "unit"
  | String_t             -> "string"
  | Func_t (args, rt)    -> "[(" ^ (String.concat ", " (List.map
                              (fun arg -> str_types arg) args)) ^ ") => " ^
                                str_types rt ^ "]"
  | List_t t             -> "list<" ^ str_cont_t t
  | Set_t t              -> "set<" ^ str_cont_t t
  | Map_t (t1, t2)       -> "map<" ^ str_types t1 ^ ", " ^ str_cont_t t2
  | Actor_t t            -> "actor<" ^ t ^ ">"
  | Pool_t t             -> "pool<" ^ t ^ ">"
  | Message_t t          -> "message<" ^ t ^ ">"

and str_types_list types =
  String.concat ", " (List.map str_types types)

and str_cont_t types =
  match types with
      List_t _ | Set_t _ | Map_t (_, _) -> str_types types ^ " >"
    | _ -> str_types types ^ ">"

and str_formal  = function
```

```
  (name, typ) -> name ^ " : " ^ str_types typ

and str_formals (formals : formal list) =
  String.concat ", " (List.map str_formal formals)

and str_message message =
  "message " ^ message.m_name ^ "(" ^

(str_formals message.m_formals) ^ ")"

and str_expr = function
    Int_Lit i                -> string_of_int i
  | Double_Lit f             -> string_of_float f
  | Char_Lit c               -> "\'" ^ Char.escaped c ^ "\'"
  | String_Lit s             -> "\"" ^ s ^ "\""
  | Bool_Lit true            -> "true"
  | Bool_Lit false           -> "false"
  | Unit_Lit _               -> "unit"
  | Id s                     -> s
  | Access (cont, it)        -> str_expr cont ^ "[" ^ str_expr it ^ "]"
  | Func_Lit fl              -> str_fl fl
  | List_Lit (t, ex)         -> "list<" ^ str_cont_t t  ^ ">[" ^
                                  str_exprs ex  ^ "]"
  | Set_Lit (t, ex)          -> "set<" ^  str_cont_t t  ^ ">[" ^
                                  str_exprs ex  ^ "]"
  | Map_Lit (kt, vt, kvs)    -> "map<" ^ str_types kt ^ ", " ^
                                 str_cont_t vt ^ ">[" ^ str_kvs kvs ^ "]"
  | Actor_Lit (at, ex)       -> "spawn actor<" ^ at ^ ">(" ^
                                  str_exprs ex ^ ")"
  | Pool_Lit (at, ex, num)   -> "spawn pool<" ^ at ^ ">({" ^
                                  str_exprs ex ^ "}, " ^
                                    str_expr num ^ ")"
  | Message_Lit (m, ex)      -> "message<" ^ m ^ ">(" ^
                                  str_exprs ex ^ ")"
  | Binop (e1, o, e2)        -> "(" ^ str_expr e1 ^ " " ^ str_binop o ^
                                  " " ^ str_expr e2 ^ ")"
  | Uop (o, e)              -> str_uop o ^ str_expr e

  | FuncCall (s, ex)        -> s ^ "(" ^ str_exprs ex ^ ")"
  | Noexpr                  -> ""

and str_exprs ex =
  String.concat ", " (List.map str_expr ex)

(* helper to print map key-value pairs *)
and str_kvs kvs =
  let kv_string kv =
    let (k, v) = kv in str_expr k ^ " -> " ^ str_expr v in
  String.concat ", " (List.map kv_string kvs)



and str_stmt = function
    Block stmts         -> "{\n" ^ str_stmts (Block(stmts)) ^ "\n}"
  | Expr expr           -> str_expr expr ^ ";"
  | Return expr         -> "return " ^ str_expr expr ^ ";"
  | Mutdecl mv          -> "mut " ^ str_types mv.mv_type ^ " " ^
                             mv.mv_name ^ (match mv.mv_init with
                                Noexpr -> ""
                              | _ -> " = " ^ str_expr mv.mv_init) ^ ";"
  | Vdecl v             -> str_vdecl v
  | If (e, s1, s2)      -> str_if e s1 s2
  | Actor_send (e, a)   -> str_expr e ^ " |> " ^ str_expr a ^ ";"
  | Pool_send (e, p)    -> str_expr e ^ " |>> " ^ str_expr p ^ ";"

and str_stmts = function
  Block(stmts)  -> String.concat "\n" (List.map str_stmt stmts)
  | _           -> ""

and str_if e s1 s2 =
  "if (" ^ str_expr e ^ ") " ^ str_stmt s1 ^ (match s2 with
      Expr Noexpr  -> ""
    | _            -> " else " ^ str_stmt s2)
```

```
and str_fl fl =
  let { f_formals = formals; f_return_t = rt; f_body = body } = fl in
  let s = match body with
      Block stmts -> stmts
    | _ -> [] in

  match (List.length s, List.hd s) with
      (1, Return e) ->
        "(" ^ str_formals formals ^ ") => " ^
        str_types rt ^ " = " ^ str_expr e
    | _              ->
        "(" ^ str_formals formals ^ ") => " ^
          str_types rt ^ " = " ^ str_stmt fl.f_body

and str_func name f =
  let { f_formals = formals; f_return_t = rt; f_body = body } = f in
  let s = match body with
      Block stmts -> stmts
    | _ -> [] in

  match (List.length s, List.hd s) with
      (1, Return e) ->
        "func " ^ name ^ " = (" ^ str_formals formals ^ ") => " ^
        str_types rt ^ " = " ^ str_expr e ^ ";"
    | _              ->
        "def " ^ name ^ "(" ^ str_formals formals ^ ") => " ^
        str_types rt ^ " = " ^ str_stmt body

and str_vdecl v =
  (match v.v_init with
    Func_Lit(f) -> str_func v.v_name f
    | _ ->
        str_types v.v_type ^ " " ^ v.v_name ^ " = " ^
            str_expr v.v_init ^ ";")
```

```
and str_pattern p =
  "| " ^ p.p_mid ^ "(" ^ str_formals p.p_mformals ^ ") => " ^
      str_stmt p.p_body

let str_actor actor =
  "actor " ^ actor.a_name ^ "(" ^ str_formals actor.a_formals ^ ") {\n" ^
    str_stmts actor.a_body ^ "\n\n" ^ "\n\nreceive = {\n" ^
      String.concat "\n" (List.map str_pattern actor.a_receive) ^ "\n}\n}"

let str_program (messages, actors, funcs) =
  String.concat "\n" (List.map str_message messages) ^ "\n\n" ^
    String.concat "\n\n" (List.map str_actor actors) ^ "\n\n" ^
      String.concat "\n\n" (List.map str_vdecl funcs
```

8.1.3 parser.mly

```
%{ open Ast %}

%token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET
%token ARITH_PLUS ARITH_MINUS ARITH_TIMES ARITH_DIVIDE ARITH_MOD
%token ASSIGN
%token PUNC_COMMA PUNC_SEMI
%token LOGIC_EQ LOGIC_NEQ LANGLE RANGLE LOGIC_LEQ LOGIC_GEQ LOGIC_NOT
%token LOGIC_AND LOGIC_OR LOGIC_TRUE LOGIC_FALSE
%token BITWISE_AND BITWISE_OR BITWISE_XOR  BITWISE_RIGHT BITWISE_LEFT
%token FUNC_ARG_TYPE ARROW FUNC_RET_TYPE RETURN
%token FLOW_IF FLOW_ELSE
%token ACT_SPAWN ACT_RECEIVE ACT_SEND ACT_BROADCAST
%token MUTABLE
%token TYPE_INT TYPE_DOUBLE TYPE_CHAR TYPE_BOOL TYPE_UNIT TYPE_STR
%token TYPE_LIST TYPE_SET TYPE_MAP
%token TYPE_MESSAGE TYPE_ACTOR TYPE_POOL FUNC_DECL TYPE_FUNC
%token <int> INT_LIT
%token <float> DOUBLE_LIT
%token <string> STRING_LIT
%token <char> CHAR_LIT
%token <bool> BOOL_LIT
%token <string> ID
%token EOF

%nonassoc NOELSE LBRACKET
%left ACT_SEND ACT_BROADCAST
%right ASSIGN
%left FUNC_ARG_TYPE ARROW FUNC_RET_TYPE
%left LOGIC_AND LOGIC_OR
%left LOGIC_EQ LOGIC_NEQ
%left LANGLE RANGLE LOGIC_LEQ LOGIC_GEQ
%left BITWISE_AND BITWISE_XOR BITWISE_OR
%right BITWISE_LEFT BITWISE_RIGHT
%left ARITH_PLUS ARITH_MINUS
```

```
%left ARITH_TIMES ARITH_DIVIDE ARITH_MOD
%right LOGIC_NOT

%right NEG /* for negative numbers */

%start program
%type <Ast.program> program

%%

program:
  messages actors functions EOF { ($1, $2, $3) }

/**********
MESSAGES
***********/

messages:
    /* nothing */  { [] }
  | message_list   { List.rev $1 }

message_list:
  message_decl                { [$1] }
  | message_list message_decl  { $2::$1 }

message_decl:
  TYPE_MESSAGE ID LPAREN formals_opt RPAREN { { m_name = $2; m_formals = $4 } }

/**********
ACTORS
***********/

actors:
    /* nothing */  { [] }
  | actor_list     { List.rev $1 }
```

```
actor_list:
  actor_decl               { [$1] }
  | actor_list actor_decl  { $2::$1 }

actor_decl:
  TYPE_ACTOR ID LPAREN formals_opt RPAREN LBRACE stmts receive RBRACE
     { { a_name = $2; a_formals = $4; a_body = $7; a_receive = $8 } }

/**********
FUNCTIONS
***********/

functions:
    /* nothing */        { [] }
  | function_list        { List.rev $1 }

function_list:
    fdecl                { [$1] }
  | function_list fdecl  { $2::$1 }

fdecl:
    FUNC_DECL ID LPAREN formals_opt RPAREN FUNC_RET_TYPE typ
      ASSIGN LBRACE stmts RBRACE
        { { v_name = $2;
            v_type = Func_t((List.map (fun (_, t) -> t) $4), $7);
            v_init = Func_Lit({ f_formals = $4; f_return_t = $7;
                                f_body = $10 }) } }
  | TYPE_FUNC ID ASSIGN LPAREN formals_opt RPAREN FUNC_RET_TYPE typ
      ASSIGN expr PUNC_SEMI
        { { v_name = $2;
            v_type = Func_t((List.map (fun (_, t) -> t) $5), $8);
            v_init = Func_Lit({ f_formals = $5; f_return_t = $8;
                                f_body = Block([Return $10]); }) } }

formals_opt:
    /* nothing */ { [] }
  | formal_list  { List.rev $1 }

formal_list:
    ID FUNC_ARG_TYPE typ                         { [($1, $3)] }
  | formal_list PUNC_COMMA ID FUNC_ARG_TYPE typ  { ($3, $5) :: $1 }

/* primative types */

typ_opt:
    /* nothing */ { [] }
  | typ_list      { List.rev $1 }

typ_list:
    typ                     { [$1] }
  | typ_list PUNC_COMMA typ { $3 :: $1 }

typ:
    simple_typ   { $1 }
  | cont_typ     { $1 }
  | actor_typ    { $1 }
  | func_typ   { $1 }
  | message_typ  { $1 }


simple_typ:
    TYPE_INT     { Int_t }
  | TYPE_BOOL    { Bool_t }
  | TYPE_DOUBLE  { Double_t }
  | TYPE_CHAR    { Char_t }
  | TYPE_UNIT    { Unit_t }

cont_typ:
    TYPE_STR                                    { String_t }
  | TYPE_MAP LANGLE typ PUNC_COMMA typ RANGLE  { Map_t($3, $5) }
  | TYPE_SET LANGLE typ RANGLE                  { Set_t($3) }
  | TYPE_LIST LANGLE typ RANGLE                 { List_t($3) }
```

```
actor_typ:
    TYPE_ACTOR LANGLE ID RANGLE  { Actor_t($3) }
  | TYPE_POOL LANGLE ID RANGLE   { Pool_t($3) }

func_typ:
  LBRACKET LPAREN typ_opt RPAREN FUNC_RET_TYPE typ RBRACKET
      { Func_t($3, $6) }

message_typ:
  TYPE_MESSAGE LANGLE ID RANGLE { Message_t($3) }

/* for pattern matching with receive */
receive:
  ACT_RECEIVE ASSIGN LBRACE pattern_opt RBRACE { $4 }

pattern_opt:
    /* nothing */ { [] }
  | pattern_list  { List.rev $1 }



pattern_list:
  pattern                { [$1] }
  | pattern_list pattern  { $2::$1 }

pattern:
  BITWISE_OR ID LPAREN formals_opt RPAREN FUNC_RET_TYPE LBRACE stmts RBRACE
    { { p_mid = $2; p_mformals = $4; p_body = $8; } }

mut_vdecl:
    MUTABLE typ ID              { Mutdecl({ mv_name = $3;
                                            mv_type = $2;
                                            mv_init = Noexpr}) }
  | MUTABLE typ ID ASSIGN expr  { Mutdecl({ mv_name = $3;
                                            mv_type = $2;
                                            mv_init = $5}) }
```

```
stmts:
    /* nothing */  { Block([]) }
  | stmt_list      { Block(List.rev $1) }

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

stmt:
    expr PUNC_SEMI                 { Expr $1 }
  | typ ID ASSIGN expr PUNC_SEMI  { Vdecl({ v_name = $2;
                                      v_type = $1;
                                      v_init = $4}) }
  | mut_vdecl PUNC_SEMI           { $1 }
  | fdecl                         { Vdecl($1) }
  | RETURN PUNC_SEMI              { Return Noexpr }
  | RETURN expr PUNC_SEMI         { Return $2 }
  | LBRACE stmts RBRACE           { $2 }
  | stmt_cond                     { $1 }
  | expr ACT_SEND ID PUNC_SEMI
                                 { Actor_send($1, Id($3)) }
  | expr ACT_BROADCAST ID PUNC_SEMI
                                 { Pool_send($1, Id($3)) }
stmt_cond:
    FLOW_IF LPAREN expr RPAREN LBRACE stmts RBRACE %prec NOELSE
                                       { If($3, $6, Expr(Noexpr)) }
  | FLOW_IF LPAREN expr RPAREN LBRACE stmts RBRACE
        FLOW_ELSE LBRACE stmts RBRACE  { If($3, $6, $10) }

map_opt:
    /* nothing */ { [] }
  | map_list      { List.rev $1 }

map_list:
    expr ARROW expr                          { [($1, $3)] }
  | map_list PUNC_COMMA expr ARROW expr  { ($3, $5) :: $1 }
```

```
cont_lit:
    TYPE_LIST LANGLE typ RANGLE
        LBRACKET actuals_opt RBRACKET            { List_Lit($3, $6) }
  | TYPE_SET LANGLE typ RANGLE
        LBRACKET actuals_opt RBRACKET            { Set_Lit($3, $6) }
  | TYPE_MAP LANGLE typ PUNC_COMMA typ RANGLE
        LBRACKET map_opt RBRACKET                { Map_Lit($3, $5, $8) }

actor_lit:
    ACT_SPAWN TYPE_ACTOR LANGLE ID RANGLE LPAREN actuals_opt RPAREN
                                 { Actor_Lit($4, $7) }
  | ACT_SPAWN TYPE_POOL LANGLE ID RANGLE LPAREN LBRACE actuals_opt RBRACE
      PUNC_COMMA expr RPAREN        { Pool_Lit($4, $8, $11) }
message_lit:
  TYPE_MESSAGE LANGLE ID RANGLE LPAREN actuals_opt RPAREN
      { Message_Lit($3, $6) }

func_lit:
  LPAREN formals_opt RPAREN FUNC_RET_TYPE typ ASSIGN LBRACE stmts RBRACE
    { Func_Lit({ f_formals = $2; f_return_t = $5; f_body = $8; }) }
  | LPAREN formals_opt RPAREN FUNC_RET_TYPE typ ASSIGN expr
    { Func_Lit({ f_formals = $2; f_return_t = $5;
          f_body = Block([Return $7]); }) }

expr:
    ID                                       { Id($1) }
  | INT_LIT                                  { Int_Lit($1) }
  | DOUBLE_LIT                               { Double_Lit($1) }
  | CHAR_LIT                                 { Char_Lit($1) }
  | STRING_LIT                               { String_Lit($1) }
  | BOOL_LIT                                 { Bool_Lit($1) }
  | TYPE_UNIT                                { Unit_Lit() }
  | LOGIC_TRUE                               { Bool_Lit(true) }
  | LOGIC_FALSE                              { Bool_Lit(false) }
  | cont_lit                                 { $1 }
  | actor_lit                                { $1 }
  | message_lit                              { $1 }
  | func_lit                                 { $1 }
  | expr ARITH_PLUS     expr                 { Binop($1, Add, $3) }
  | expr ARITH_MINUS    expr                 { Binop($1, Sub, $3) }
  | expr ARITH_TIMES    expr                 { Binop($1, Mult, $3) }
  | expr ARITH_DIVIDE   expr                 { Binop($1, Div, $3) }
  | expr ARITH_MOD      expr                 { Binop($1, Mod, $3) }
  | expr LOGIC_EQ       expr                 { Binop($1, Equal, $3) }
  | expr LOGIC_NEQ      expr                 { Binop($1, Neq, $3) }
  | expr LANGLE         expr                 { Binop($1, Less, $3) }
  | expr LOGIC_LEQ      expr                 { Binop($1, Leq, $3) }
  | expr RANGLE         expr                 { Binop($1, Greater,$3) }
  | expr LOGIC_GEQ      expr                 { Binop($1, Geq, $3) }
  | expr LOGIC_AND      expr                 { Binop($1, And, $3) }
  | expr LOGIC_OR       expr                 { Binop($1, Or, $3) }
  | expr BITWISE_AND    expr                 { Binop($1, Bit_And, $3) }
  | expr BITWISE_OR     expr                 { Binop($1, Bit_Or, $3) }
  | expr BITWISE_XOR    expr                 { Binop($1, Bit_Xor, $3) }
  | expr BITWISE_RIGHT  expr                 { Binop($1, Bit_RShift, $3) }
  | expr BITWISE_LEFT   expr                 { Binop($1, Bit_LShift, $3) }
  | ARITH_MINUS expr %prec NEG               { Uop(Neg, $2) }
  | LOGIC_NOT expr                           { Uop(Not, $2) }
  | ID LPAREN actuals_opt RPAREN             { FuncCall($1, $3) }
  | LPAREN expr RPAREN                       { $2 }
  | expr ASSIGN expr                         { Binop($1, Assign, $3) }
  | expr LBRACKET expr RBRACKET              { Access($1, $3) }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                           { [$1] }
  | actuals_list PUNC_COMMA expr  { $3 :: $1 }
```

8.1.4 sast.ml

```
open Ast

type sexpr =
    SInt_Lit      of int
  | SDouble_Lit   of float
  | SChar_Lit     of char
  | SString_Lit   of string
  | SBool_Lit     of bool
  | SUnit_Lit     of unit
  | SId           of string
  | SAccess       of t_expr * t_expr
  | SFunc_Lit     of sfunc
  | SList_Lit     of types * t_expr list
  | SSet_Lit      of types * t_expr list
  | SMap_Lit      of types * types * (t_expr * t_expr) list
  | SActor_Lit    of string * (t_expr list)
  | SPool_Lit     of string * (t_expr list) * t_expr
  | SMessage_Lit  of string * (t_expr list)
  | SBinop        of t_expr * bin_op * t_expr
  | SUop          of u_op * t_expr
  | SFuncCall     of string * (t_expr list)
  | SNoexpr

and sstmt =
    SBlock       of sstmt list
  | SExpr        of t_expr
  | SReturn      of t_expr
  | SVdecl       of sval_decl
  | SMutdecl     of smvar_decl
  | SIf          of t_expr * sstmt * sstmt
  | SActor_send  of t_expr * t_expr
  | SPool_send   of t_expr * t_expr

and t_expr = sexpr * types
and sval_decl = {
    sv_name : string;
    sv_type : types;
    sv_init : t_expr;
}

and smvar_decl = {
    smv_name : string;
    smv_type : types;
    smv_init : t_expr;
}

and sfunc = {
  sf_formals  : formal list;
  sf_return_t : types;
  sf_body     : sstmt;
}

and smessage = {
  sm_name    : string;
  sm_formals : formal list;
}

and spattern = {
  sp_smid      : string;
  sp_smformals : formal list;
  sp_body      : sstmt;
}

and sactor = {
  sa_name      : string;
  sa_formals   : formal list;
  sa_body      : sstmt;
  sa_receive   : spattern list;
}
```

```
and vsymtab = {
  vparent      : vsymtab option;
  svals        : sval_decl list
}

and mvsymtab = {
  mvparent     : mvsymtab option;
  smvars       : smvar_decl list
}

and actor_scope = {
  a_actor    : sactor;
  a_scope    : scope;
  a_messages : smessage list;
}

and scope = {
  messages    : smessage list;
  actors      : actor_scope list;
  env_vtable  : vsymtab;
  env_mvtable : mvsymtab;
  return_t    : types option;
  in_actor    : bool;
  actor_init    : bool;
}

type sprogram = smessage list * actor_scope list * sval_decl list * sfunc

(* ========== *)

let str_smessage smessage =
  "message " ^ smessage.sm_name ^ "(" ^ (str_formals smessage.sm_formals) ^ ")"

let rec str_texpr texpr =
  (match (fst texpr) with
    SInt_Lit i                -> string_of_int i
  | SDouble_Lit f             -> string_of_float f
  | SChar_Lit c               -> "\'" ^ Char.escaped c ^ "\'"
  | SString_Lit s             -> "\"" ^ s ^ "\""
  | SBool_Lit true            -> "true"
  | SBool_Lit false           -> "false"
  | SUnit_Lit _               -> "unit"
  | SId se                    -> se
  | SAccess (scont, sit)      -> str_texpr scont ^ "[" ^ str_texpr sit ^ "]"
  | SFunc_Lit sfl             -> str_sfl sfl
  | SList_Lit (t, sel)        -> "list<" ^ str_cont_t t ^ "[" ^
                                   str_texprs sel ^ "]"
  | SSet_Lit (t, sel)         -> "set<" ^ (str_cont_t t) ^ "[" ^
                                   str_texprs sel ^ "]"
  | SMap_Lit (kt, vt, skvs)   -> "map<" ^ str_types kt ^ ", " ^
                                   str_cont_t vt ^ "[" ^
                                     str_skvs skvs ^ "]"
  | SActor_Lit (sat, sel)     -> "spawn actor<" ^ sat ^ ">(" ^
                                   str_texprs sel ^ ")"
  | SPool_Lit (sat, sel, num)  -> "spawn pool<" ^ sat ^ ">({" ^
                                   str_texprs sel ^ "}, " ^
                                     str_texpr num ^ ")"
  | SMessage_Lit (m, sel)     -> "message<" ^ m ^ ">(" ^
                                   str_texprs sel ^ ")"
  | SBinop (se1, o, se2)      -> "(" ^ str_texpr se1 ^ " " ^ str_binop o
                                     ^ " " ^ str_texpr se2 ^ ")"
  | SUop (o, se)              -> str_uop o ^ str_texpr se
  | SFuncCall (se, sel)       -> se ^ "(" ^ str_texprs sel ^ ")"
  | SNoexpr                   -> ""
  ) (*^ "{{TYPE: "^ str_types (snd texpr) ^ "}}" *)

and str_texprs sel =
  String.concat ", " (List.map str_texpr sel)

and str_skvs skvs =
  let skv_string skv =
    let (sk, sv) = skv in str_texpr sk ^ " -> " ^ str_texpr sv in
  String.concat ", " (List.map skv_string skvs)

and str_sstmt = function
    SBlock(sstmts)       -> "{\n" ^ str_sstmts (SBlock(sstmts)) ^ "\n}"
  | SExpr texp           -> str_texpr texp ^ ";"
  | SReturn(se)          -> "return " ^ str_texpr se ^ ";"
  | SMutdecl smv         -> "mut " ^ (str_types smv.smv_type) ^ " " ^
                              smv.smv_name ^ (match smv.smv_init with
                                  SNoexpr, _ -> ""
                                | _ -> " = " ^ (str_texpr smv.smv_init)) ^ ";"
  | SVdecl sv            -> str_svdecl sv
  | SIf (se, s1, s2)     -> str_sif se s1 s2
  | SActor_send (se, a)  -> str_texpr se ^ " |> " ^ str_texpr a ^ ";"
  | SPool_send (se, p)   -> str_texpr se ^ " |>> " ^ str_texpr p ^ ";"

and str_sstmts = function
    SBlock(sstmts)  -> String.concat "\n" (List.map str_sstmt sstmts)
  | _           -> ""

and str_sif se s1 s2 =
  "if (" ^ str_texpr se ^ ") " ^ str_sstmt s1 ^ (match s2 with
      SExpr (SNoexpr, _)  -> ""
    | _  -> " else " ^ str_sstmt s2)

and str_sfl sfl =
  let { sf_formals = sformals; sf_return_t = srt; sf_body = sbody } = sfl in
  let s = match sbody with
      SBlock sstmts -> sstmts
    | _ -> [] in

  match (List.length s, List.hd s) with
      (1, SReturn te) ->
        "(" ^ str_formals sformals ^ ") => " ^
        str_types srt ^ " = " ^ str_texpr te
    | _                ->
```

```
        "(" ^ str_formals sformals ^ ") => " ^
          str_types srt ^ " = " ^ str_sstmt sfl.sf_body

and str_svdecl sv =
  (match sv.sv_init with
    (SFunc_Lit(sf), _) -> str_sfunc sv.sv_name sf
    | _ ->
        str_types sv.sv_type ^ " " ^ sv.sv_name ^ " = "
        ^ str_texpr sv.sv_init ^ ";")

and str_sfunc name sf =
  let { sf_formals = sformals; sf_return_t = srt; sf_body = sbody } = sf in
  let s = match sbody with
      SBlock sstmts -> sstmts
    | _ -> [] in

  match (List.length s, List.hd s) with
      (1, SReturn te) ->
        "func " ^ name ^ " = (" ^ str_formals sformals ^ ") => " ^
        str_types srt ^ " = " ^ str_texpr te ^ ";"
    | _               ->
        "def " ^ name ^ "(" ^ str_formals sformals ^ ") => " ^
        str_types srt ^ " = " ^ str_sstmt (SBlock s)

and str_spattern sp =
  "| " ^ sp.sp_smid ^ "(" ^ str_formals sp.sp_smformals ^ ") => " ^
    str_sstmt sp.sp_body

let str_sactor sactor_scope =
  let sactor = sactor_scope.a_actor in
  "actor " ^ sactor.sa_name ^ "(" ^ str_formals sactor.sa_formals ^ ") {\n" ^
    str_sstmts sactor.sa_body ^ "\n\n\n\nreceive = {\n" ^ String.concat "\n"
      (List.map str_spattern sactor.sa_receive) ^ "\n}\n}"

let str_sprogram (smessages, sactors, sfuncs, main) =
  String.concat "\n" (List.map str_smessage smessages) ^ "\n\n" ^
  String.concat "\n\n" (List.map str_sactor sactors) ^ "\n\n" ^
  String.concat "\n\n" (List.map str_svdecl sfuncs) ^ "\n\n" ^
  (match main with
      (SFunc_Lit(sf), _) -> str_sfunc "main" sf
    | _                  -> "")
```

8.1.5 Analyzer.ml

```
open Ast
open Sast

exception Duplicate_decl
exception Type_mismatch
exception Builtin_arg_num_err of string * int * int
exception Builtin_arg_type_err of string * (t_expr list)

let get_list_fst l  =
  List.map (fun (x, _) -> x) l

let get_list_snd l =
  List.map (fun (_, t) -> t) l

let empty_fbody = SExpr (SNoexpr, Unit_t)

let upd_vtable_vals (nsval : sval_decl) (v_table : vsymtab) =
  { v_table with svals = nsval :: v_table.svals }

let upd_mvtable_vals (nmsvar : smvar_decl) (mv_table : mvsymtab) =
  { mv_table with smvars = nmsvar :: mv_table.smvars }

let upd_env_vtable (vtab : vsymtab) (env : scope) =
  { env with env_vtable = vtab }

let upd_env_mvtable (mvtab : mvsymtab) (env : scope) =
  { env with env_mvtable = mvtab }

let upd_env_messages (m: smessage) (env : scope) =
  { env with messages = m :: env.messages }

let rec find_value_decl (v_name : string) (vstab : vsymtab) =
  try
```

```
    List.find (fun sval -> sval.sv_name = v_name) vstab.svals
  with Not_found ->
    match vstab.vparent with
        Some(vparent) -> find_value_decl v_name vparent
      | _ -> raise Not_found

let rec find_variable_decl (mv_name : string) (mvstab : mvsymtab) =
  try
    List.find (fun mvar -> mvar.smv_name = mv_name) mvstab.smvars
  with Not_found ->
    match mvstab.mvparent with
        Some mvparent -> find_variable_decl mv_name mvparent
      | _ -> raise Not_found

let is_immu (v_name : string) (vstab : vsymtab) =
  try
    let _ = find_value_decl v_name vstab in true
  with
     Not_found -> false

let is_mut (mv_name : string) (mvstab: mvsymtab) =
  try
    let _ = find_variable_decl mv_name mvstab in true
  with
    Not_found -> false

let find_vtype (name : string) (env : scope) =
  let vstab = env.env_vtable and mvstab = env.env_mvtable in
    try
      let vtype = find_value_decl name vstab in
      try
        let mvtype = find_variable_decl name mvstab in mvtype.smv_type
      with
          Not_found -> vtype.sv_type
        | _ -> raise Duplicate_decl
    with Not_found ->
```

```
      let mvtype = find_variable_decl name mvstab in mvtype.smv_type

let is_type_apm (t : types) =
  match t with
      Actor_t _ | Pool_t _ | Message_t _ -> true
    | _ -> false

let validate_arg_types (args : types list) =
  try
    List.fold_left (fun acc arg_t ->
       acc && not (is_type_apm arg_t)
    ) true args
  with Not_found -> true

let rec types_equal (t1 : types) (t2 : types) =
  match t1, t2 with
      Int_t, Int_t       -> true
    | Double_t, Double_t -> true
    | Char_t, Char_t     -> true
    | String_t, String_t -> true
    | Bool_t, Bool_t     -> true
    | Unit_t, Unit_t     -> true
    | Func_t (fl1, rt1), Func_t (fl2, rt2) ->
        check_args_t fl1 fl2 && types_equal rt1 rt2
    | List_t lt1, List_t lt2 when types_equal lt1 lt2 -> true
    | Set_t st1, Set_t st2 when types_equal st1 st2 -> true
    | Map_t (kt1, vt1), Map_t (kt2, vt2) when
        types_equal kt1 kt2 && types_equal vt1 vt2 -> true
    | _ -> false

and check_args_t (params : types list) (args : types list) =
  if not (List.length params = List.length args) then
    false
  else
    List.fold_left2 (fun acc p1 p2 -> acc && types_equal' p1 p2) true params args
```

```
and types_equal' (t1 : types) (t2 : types) =
  types_equal t1 t2 || (
    match t1, t2 with
        Message_t mt1, Message_t mt2 when mt1 = mt2 -> true
      | Actor_t at1, Actor_t at2 when at1 = at2 -> true
      | Pool_t pt1, Pool_t pt2 when pt1 = pt2 -> true
      | _ -> false
  )

let check_args (params : types list) (args : t_expr list) =
  check_args_t params (get_list_snd args)

let find_message (sm_name : string) (env : scope) =
  try
    List.find (fun sm -> sm_name = sm.sm_name) env.messages
  with Not_found ->
    raise (Failure ("Message of type " ^ sm_name ^ " not found"))

let find_actor_scope (sa_name : string) (env : scope) =
  try
    let actor_scope = List.find (fun a_scope ->
      sa_name = a_scope.a_actor.sa_name
    ) env.actors in actor_scope
  with Not_found ->
    raise (Failure ("Actor of type " ^ sa_name ^ " not found"))

let name_taken (name : string) (env : scope) =
    try
      let _ = find_vtype name env in true
    with Not_found -> false

let find_actor (sa_name : string) (env : scope) =
  let actor_scope = find_actor_scope sa_name env in
  actor_scope.a_actor

let check_message_lit (sm: smessage) (args : t_expr list) =
  let req_params = get_list_snd sm.sm_formals in
  if check_args req_params args then
    (SMessage_Lit (sm.sm_name, args), Message_t sm.sm_name)
  else
    raise (Failure ("Message constructed with conflicting parameter types " ^
      sm.sm_name ^ " requires (" ^ str_types_list req_params ^
        ") but constructed with " ^ str_types_list (get_list_snd args)))

let check_actor_lit (sa : sactor) (args : t_expr list) =
  let req_params = get_list_snd sa.sa_formals in
  if check_args req_params args then
    (SActor_Lit (sa.sa_name, args), Actor_t sa.sa_name)
  else
    raise (Failure ("Actor constructed with conflicting parameter types " ^
      sa.sa_name ^ " requires (" ^ str_types_list req_params ^
        ") but constructed with " ^ str_types_list (get_list_snd args)))


let check_builtin (f : string) (tel : t_expr list) (env : scope) =
  let args_len = List.length tel in
    match f with
        "Println" ->
          (match tel with
              [(_, tail_t)] ->
                (match tail_t with
                    Actor_t _ | Pool_t _ | Message_t _ ->
                      raise (Builtin_arg_type_err (f, tel))
                  | _ -> (SFuncCall(f, tel), Unit_t))
            | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
      | "Die" ->
          (if env.in_actor then
            if args_len > 0 then
              raise (Builtin_arg_num_err (f, 0, args_len))
            else
              (SFuncCall("Die", []), Unit_t)
          else
            raise (Failure "Die() can only be called in actors"))
      | "AsInt" ->
          (match tel with
            [(_, tail_t)] ->
                (match tail_t with
                    Double_t -> (SFuncCall(f, tel), Int_t)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
      | "AsDouble" ->
          (match tel with
            [(_, tail_t)] ->
                (match tail_t with
                    Int_t -> (SFuncCall(f, tel), Double_t)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
      | "AsString" ->
          (match tel with
            [(_, tail_t)] ->
                (match tail_t with
                    Actor_t _ | Pool_t _ | Message_t _ ->
                      raise (Builtin_arg_type_err (f, tel))
                  | _ -> (SFuncCall(f, tel), String_t))
            | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
      | "Reverse" | "PopFront" | "PopBack" ->
          (match tel with
              [(_, tail_t)] ->
                (match tail_t with
                    List_t _ -> (SFuncCall(f, tel), tail_t)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
      | "Size" ->
          (match tel with
              [(_, tail_t)] ->
                (match tail_t with
                    String_t | List_t _ | Set_t _ | Map_t (_, _) ->
                      (SFuncCall(f, tel), Int_t)
```

```
           | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 1, args_len)))
| "Contains" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match tail_t with
              List_t ct | Set_t ct | Map_t (ct, _) when
                types_equal head_t ct -> (SFuncCall(f, tel), Bool_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "Prepend" | "Append" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match tail_t with
              List_t ct when
                types_equal head_t ct -> (SFuncCall(f, tel), tail_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "MergeFront" | "MergeBack"->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match head_t, tail_t with
              List_t lt1, List_t lt2 when
                types_equal lt1 lt2 -> (SFuncCall(f, tel), tail_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "Union" | "Diff" | "Intersection" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match head_t, tail_t with
              Set_t st1, Set_t st2 when
                types_equal st1 st2 -> (SFuncCall(f, tel), tail_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "Put" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match tail_t with
              Set_t ct when
                types_equal head_t ct -> (SFuncCall(f, tel), tail_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | (_, head_t) :: [(_, v_t) ; (_, tail_t)] ->
          (match tail_t with
              Map_t (kt, vt) when
                types_equal head_t kt && types_equal v_t vt ->
                  (SFuncCall(f, tel), tail_t)
            | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "ForEach" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match head_t, tail_t with
            Func_t ([ft], rt), (List_t ct | Set_t ct) when
              types_equal ft ct && types_equal rt Unit_t ->
                (SFuncCall(f, tel), Unit_t)
          | Func_t (fkt :: [fvt], rt), Map_t (kt, vt) when
              types_equal fkt kt && types_equal fvt vt &&
                types_equal rt Unit_t -> (SFuncCall (f, tel), Unit_t)
          | _ -> raise (Builtin_arg_type_err (f, tel)))
      | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
| "Map" ->
    (match tel with
        (_, head_t) :: [(_, tail_t)] ->
          (match head_t, tail_t with
              Func_t ([ft], rt), List_t lt when types_equal ft lt ->
                (SFuncCall (f, tel), List_t rt)
            | Func_t ([ft], rt), Set_t st when types_equal ft st ->
                (SFuncCall (f, tel), Set_t rt)
            | Func_t (fkt :: [fvt], rt), Map_t (kt, vt) when
                types_equal fkt kt && types_equal fvt vt ->
                  (SFuncCall (f, tel), Map_t (kt, rt))
            | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
      | "Filter" ->
          (match tel with
              (_, head_t) :: [(_, tail_t)] ->
                (match head_t, tail_t with
                    Func_t ([ft], rt), (List_t ct | Set_t ct) when
                      types_equal ft ct && types_equal rt Bool_t ->
                        (SFuncCall (f, tel), tail_t)
                  | Func_t (fkt :: [fvt], rt), Map_t (kt, vt) when
                      types_equal fkt kt && types_equal fvt vt &&
                        types_equal rt Bool_t -> (SFuncCall (f, tel), tail_t)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
      | "FoldLeft" ->
          (match tel with
              (_, head_t) :: [(_, acc) ; (_, tail_t)] ->
                (match head_t, tail_t with
                    Func_t (ft :: [acct], rt), (List_t ct | Set_t ct) when
                      types_equal ft ct && types_equal acc acct &&
                        types_equal acc rt -> (SFuncCall(f, tel), rt)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 3, args_len)))
      | "Reduce" ->
          (match tel with
              (_, head_t) :: [(_, tail_t)] ->
                (match head_t, tail_t with
                    Func_t (ft :: [acc], rt), (List_t ct | Set_t ct) when
                      types_equal ft ct && types_equal acc rt ->
                        (SFuncCall(f, tel), rt)
                  | _ -> raise (Builtin_arg_type_err (f, tel)))
            | _ -> raise (Builtin_arg_num_err (f, 2, args_len)))
      | _ -> raise Not_found

let check_binop (te1 : t_expr) (te2 : t_expr)
    (op : bin_op) (env : scope) =
  let (e1, t1) = te1 and (_, t2) = te2 in
```

```
match op with
    Add ->
      (match t1, t2 with
          Int_t, Int_t       -> (SBinop (te1, op, te2), Int_t)
        | Double_t, Double_t -> (SBinop (te1, op, te2), Double_t)
        | String_t, String_t -> (SBinop (te1, op, te2), String_t)
        | _ -> raise (Failure ("operand type mismatch: " ^
                 (str_types t1) ^ " " ^ str_binop op ^ " " ^
                   (str_types t2))))
  | Sub | Mult | Div ->
      (match t1, t2 with
          Int_t, Int_t       -> (SBinop (te1, op, te2), Int_t)
        | Double_t, Double_t -> (SBinop (te1, op, te2), Double_t)
        | _ -> raise (Failure ("operand type mismatch: " ^
                 (str_types t1) ^ " " ^ str_binop op ^ " " ^
                   (str_types t2))))
  | Less | Leq | Greater | Geq ->
      (match t1, t2 with
          Int_t, Int_t       -> (SBinop (te1, op, te2), Bool_t)
        | Double_t, Double_t -> (SBinop (te1, op, te2), Bool_t)
        | _ -> raise (Failure ("operand type mismatch: " ^
                 (str_types t1) ^ " " ^ str_binop op ^ " " ^
                   (str_types t2))))
  | Mod | Bit_And | Bit_Or | Bit_Xor | Bit_RShift | Bit_LShift ->
      (match t1, t2 with
          Int_t, Int_t  -> (SBinop (te1, op, te2), Int_t)
        | _ -> raise (Failure ("operand type mismatch: " ^
                 (str_types t1) ^ " " ^ str_binop op ^ " " ^
                   (str_types t2))))
  | Equal | Neq ->
      (match t1, t2 with
        Func_t (_, _), Func_t (_, _) ->
          raise (Failure "Functions cannot be compared for equality")
        | Actor_t _, Actor_t _ ->
          raise (Failure "Actors cannot be compared for equality")
        | Pool_t _, Pool_t _ ->
```

```
          raise (Failure "Pools cannot be compared for equality")
      | Message_t _, Message_t _ ->
          raise (Failure "Messages cannot be compared for equality")
      | _ ->
        (if types_equal t1 t2 then
          (SBinop(te1, op, te2), Bool_t)
        else
          raise (Failure ("Cannot compare " ^ str_types t1 ^
                      " with " ^ str_types t2))))
| And | Or ->
    (match t1, t2 with
        Bool_t, Bool_t -> (SBinop (te1, op, te2), Bool_t)
      | _ -> raise (Failure ("Only boolean expressions are allowed for " ^
              str_binop op)))
| Assign ->
    let (s, lhst) = (match e1 with
        SId s -> (s, t1)
      | SAccess ((SId s, at), _) -> (s, at)
      | _ -> raise (Failure ("Cannot assign " ^ str_types t1 ^  " as " ^
              str_types t2))
      ) in
    if (is_mut s env.env_mvtable) then
      (match t1 with
          Actor_t _ ->
            raise (Failure "Actors cannot be reassigned")
        | Pool_t _  ->
            raise (Failure "Pools cannot be reassigned")
        | Message_t _->
            raise (Failure "Messages cannot be reassigned")
        | Func_t (_, _) ->
            raise (Failure "Functions cannot be reassigned")
        | _ ->
          if types_equal t1 t2 then
            (SBinop(te1, op, te2), lhst)
          else
            raise (Failure ("Assignment to incompatible " ^
                          "types: " ^ str_types t2 ^ " cannot be " ^
                            "assigned to " ^ str_types t1))
              )
            else if is_immu s env.env_vtable then
              raise (Failure ("Reassignment to a value " ^ s))
            else
              raise (Failure ("Identifier not found: " ^ s))

let check_uop (te : t_expr) (op : u_op) =
  let (_, t) = te in match op with
      Neg ->
        (match t with
            Int_t    -> (SUop (op, te), Int_t)
          | Double_t -> (SUop (op, te), Double_t)
          | _ -> raise (Failure ("operand type mismatch: " ^ str_uop op ^
                    " on " ^ str_types t)))
    | Not ->
        (match t with
            Bool_t -> (SUop (op, te), Bool_t)
          | _ -> raise (Failure ("operand type mismatch: " ^ str_uop op ^
                    " on " ^ str_types t)))

let spread_arg (args : formal list) (vals : sval_decl list) =
  let rec gen_temp_name (len : int) =
    (match len with
        1               -> [ Char.escaped (Char.chr (96 + len)) ]
      | _ when len < 26 ->
            Char.escaped (Char.chr (96 + len)) :: gen_temp_name (len - 1)
      | _               -> raise (Failure "Too many arguments for this lambda")
    ) in

  let gen_f_formals (f_typs : types list) =
    let temp_names = gen_temp_name (List.length f_typs) in
    List.fold_left2 (fun acc n f -> (n, f) :: acc) [] temp_names f_typs in

  List.fold_left (fun acc form ->
    let (formal_name, formal_type) = form in
      let init = (match formal_type with
        Func_t (pts, rt) ->
          let tmp_sfunc = {
            sf_formals = gen_f_formals pts;
            sf_return_t = rt;
            sf_body = empty_fbody
            } in (SFunc_Lit(tmp_sfunc), formal_type)
        | _ -> (SNoexpr, formal_type)) in
      let n_vals = {
        sv_name = formal_name;
        sv_type = formal_type;
        sv_init = init
      } in n_vals :: acc
  ) vals args

let rec check_expr (e : expr) (env : scope) =
  let type_consistent (typ : types) (tel : types list) (cont : string) =
    try
      (let _ = List.find (fun t ->
        not (types_equal typ t)
      ) tel in
      raise (Failure (cont ^ " of type " ^ str_types typ ^ " cannot be " ^
        "initialized with parameters of type[s]" ^ str_types_list tel)))
    with Not_found -> () in

  match e with
      Int_Lit i           -> (SInt_Lit i, Int_t)
    | Double_Lit d        -> (SDouble_Lit d, Double_t)
    | Char_Lit c          -> (SChar_Lit c, Char_t)
    | String_Lit s        -> (SString_Lit s, String_t)
    | Bool_Lit b          -> (SBool_Lit b, Bool_t)
    | Unit_Lit u          -> (SUnit_Lit u, Unit_t)
    | Id id ->
        (try
          let v_t = find_vtype id env in
            (SId id, v_t)
        with
            Not_found ->
              raise (Failure ("Undeclared identifier " ^ id))
          | Duplicate_decl ->
              raise (Failure (id ^ " declared as both mutable and immutable")))
    | Access(e1, e2) ->
        let texp1 = check_expr e1 env and texp2 = check_expr e2 env in
        let (_, t1) = texp1 and (_, t2) = texp2 in
          (match t1, t2 with
              String_t, Int_t -> (SAccess(texp1, texp2), Char_t)
            | List_t lt, Int_t -> (SAccess(texp1, texp2), lt)
            | Set_t st, _ when types_equal st t2 -> (SAccess(texp1, texp2), st)
            | Map_t (kt, vt), _ when types_equal kt t2 ->
                                  (SAccess(texp1, texp2), vt)
            | _ -> raise (Failure ("Invalid access types " ^ str_types t1 ^
                     " cannot be accessed by " ^ str_types t2)))
    | Func_Lit fl ->
        let {f_formals; f_return_t; f_body} = fl in
        let p_types = get_list_snd f_formals in
        let () =
          (if (not (validate_arg_types p_types) || is_type_apm f_return_t) then
            raise (Failure ("Invalid function parameter/return type: " ^
              "Actors, pools and messages cannot be used as function " ^
                "arguments or return types"))
          else
            () ) in
        let nvals = spread_arg f_formals [] in
        let nv_table = { vparent = Some env.env_vtable; svals = nvals } in
        let nenv = { env with
          env_vtable = nv_table;
          return_t = Some f_return_t
        } in
        let (cfbody, _) = check_stmt f_body nenv in
        let sfunc = SFunc_Lit {
          sf_formals  = f_formals;
          sf_return_t = f_return_t;
          sf_body     = cfbody
        } in (sfunc, Func_t (get_list_snd f_formals, f_return_t))
    | List_Lit (lt, ex) ->
        let te_list = check_expr_list ex env in
        let te_list_types = get_list_snd te_list in
        let _ = type_consistent lt te_list_types "List" in
          (SList_Lit (lt, te_list), List_t lt)
    | Set_Lit (st, ex) ->
        let te_list = check_expr_list ex env in
        let te_list_types = get_list_snd te_list in
        let _ = type_consistent st te_list_types "Set" in
          (SSet_Lit (st, te_list), Set_t st)
    | Map_Lit (kt, vt, kvx) ->
        let tkv_list = List.map (fun (k, v) ->
          ((check_expr k env), (check_expr v env))
        ) kvx in
          (SMap_Lit (kt, vt, tkv_list), Map_t (kt, vt))
    | Binop (e1, op, e2) ->
        let checked_e1 = check_expr e1 env and checked_e2 = check_expr e2 env in
          check_binop checked_e1 checked_e2 op env
    | Uop (op, e) ->
        let checked_e = check_expr e env in check_uop checked_e op
    | FuncCall (f, args) ->
        let () = if (f = "main") then
          raise (Failure ("Cannot call main function") )
        else
          () in

        let checked_args = check_expr_list args env in
        let arg_types = get_list_snd checked_args in
        (if validate_arg_types arg_types then
          try
            check_builtin f checked_args env
          with
              Builtin_arg_num_err (b, e, num) ->
```

```
           raise (Failure ("Builtin function " ^ b ^ " called with too " ^
             "many args: Expected " ^ string_of_int e ^ " but got " ^
               string_of_int num))
       | Builtin_arg_type_err (b, _) ->
           raise (Failure ("Builtin function " ^ b ^ " cannot be called " ^
             "with arguments of type " ^ str_types_list arg_types))
       | Not_found ->
           (try

             let (fts, rt) =
               let v =
                 let decl = find_value_decl f env.env_vtable in
               decl.sv_init in
               (match (snd v) with
                   Func_t (fts, rt) -> (fts, rt)
                 | _ ->
                     raise (Failure ("Attempting to call " ^
                       "invalid type: " ^ f ^ " is type " ^
                       str_types (snd v)))) in

             if check_args fts checked_args then
               (SFuncCall(f, checked_args), rt)
             else
               raise (Failure ("Function " ^ f ^ " has signature (" ^
                 str_types_list fts ^ ") => " ^
                   str_types (match env.return_t with
                       Some rt -> rt
                     | None    -> Unit_t
                   ) ^ " but was called with args (" ^
                      str_types_list arg_types ^ ")"))
           with Not_found ->
                 raise (Failure ("Function " ^ f ^ " not found")))
   else
     raise (Failure ("Actors, pools and messages cannot be passed into " ^
       "functions as arguments")))
| Noexpr ->
```

```
         (SNoexpr, Unit_t)


and check_expr_list (el : expr list) (env : scope) =
    (List.map (fun ex -> check_expr ex env) el)

and check_stmt (s : stmt) (env : scope) =
  match s with
      Block sl ->
        let nvtable = { vparent = Some env.env_vtable; svals = []; } in
        let nmv_table = { mvparent = Some env.env_mvtable; smvars  = []; } in
        let nenv = upd_env_vtable nvtable (upd_env_mvtable nmv_table env) in
        let must_return =
          (match nenv.return_t with
              Some rt when rt != Unit_t -> true
            | _       -> false
          ) in
        let (checked_stmts, cnenv) = check_stmt_list sl must_return nenv in
          (SBlock checked_stmts, if env.actor_init then cnenv else env)
    | Expr e   -> (SExpr (check_expr e env), env)
    | Return e ->
        let texp = check_expr e env in
          (match env.return_t with
              Some rt ->
                if (types_equal (snd texp) rt) then
                  (SReturn texp, env)
                else
                  raise (Failure ("Return type mismatch: expected " ^
                    str_types rt ^ " but returns " ^ str_types (snd texp)))
            | _ -> raise (Failure "This function needs to return"))
    | Vdecl vdecl    -> check_vdecl vdecl env
    | Mutdecl mvdecl -> check_mvdecl mvdecl env
    | If (cond, isl, esl) ->
        let texp = check_expr cond env in
          (match snd texp with
              Bool_t ->
                let (check_if, _) = check_stmt isl env in
                let (check_esl, _) = check_stmt esl env in
                  (SIf(texp, check_if, check_esl), env)
            | _ -> raise (Failure "Condition must be a boolean"))
    | Actor_send (e1, e2)->
        let texp1 = check_expr e1 env and texp2 = check_expr e2 env in
        let (_, st1) = texp1 and (se2, st2) = texp2 in
          (match st1, st2 with
              Message_t _, Unit_t when (
                match se2 with
                    SId "sender" -> true
                  | _ -> false)  -> (SActor_send (texp1, texp2), env)
            | Message_t m_name, Actor_t a_name ->
                let sm = find_message m_name env in
                let sm_allowed = (find_actor_scope a_name env).a_messages in
                  (try
                    let _ = List.find (fun allowed_m ->
                      allowed_m.sm_name = sm.sm_name
                    ) sm_allowed in
                    (SActor_send (texp1, texp2), env)
                  with Not_found ->
                    raise (Failure ("No matching pattern to receive message " ^
                      m_name ^ " in actor " ^ a_name)))
            | _ -> raise (Failure ("Invalid send operation between " ^
                     str_types st1 ^ " to " ^ str_types st2)))
    | Pool_send (e1, e2) ->
        let texp1 = check_expr e1 env and texp2 = check_expr e2 env in
        let (_, st1) = texp1 and (se2, st2) = texp2 in
          (match st1, st2 with
              Message_t _, Unit_t when (
                match se2 with
                    SId "sender" -> true
                  | _ -> false)  ->
                      raise (Failure ("Messages cannot be broadcasted to " ^
                        "sender refs"))
            | Message_t m_name, Pool_t a_name ->
            let sm = find_message m_name env in
            let sm_allowed = (find_actor_scope a_name env).a_messages in
             (try
              let _ = List.find (fun allowed_m ->
               allowed_m.sm_name = sm.sm_name
              ) sm_allowed in
              (SPool_send (texp1, texp2), env)
             with Not_found ->
              raise (Failure ("No matching pattern to receive " ^
               "message " ^ m_name ^ " in pool of actor type " ^
                a_name)))
          | _ -> raise (Failure ("Invalid broadcast operation between " ^
               str_types st1 ^ " to " ^ str_types st2)))


    and check_vdecl (vdecl : val_decl) (env : scope) =
     let {v_name; v_type; v_init} = vdecl in
     let () = (match v_type with
       Unit_t ->
        raise (Failure ("Cannot declare value of type unit to " ^ v_name))
      | _ -> () ) in
     let () = (if (name_taken v_name env) then
       raise (Failure ("Value " ^ v_name ^ " declared already"))
      else
       () ) in
     let texp =
      let venv =
       let tmp_sv = {
        sv_name = v_name;
        sv_type = v_type;
        sv_init = (SNoexpr, v_type)
       } in
       let n_vtable = { vparent = Some env.env_vtable; svals = tmp_sv :: []} in
       { env with env_vtable = n_vtable }
      in
      check_expr v_init venv
      in

      let (se, t) = texp in
        match se with
            SNoexpr -> raise (Failure ("Must initialize value " ^ v_name))
          | _ ->
              if types_equal' t v_type then
                let svdecl = {
                  sv_name = v_name;
                  sv_type = v_type;
                  sv_init = texp
                } in
                let nv_table = upd_vtable_vals svdecl env.env_vtable in
                  (SVdecl svdecl, upd_env_vtable nv_table env)
              else
                raise (Failure ("Value initialization type mismatch: " ^
                  v_name ^ " is " ^ (str_types v_type) ^ " but " ^
                   "initialized as " ^ (str_types t)))


    and check_mvdecl (mvdecl : mvar_decl) (env : scope) =
      if env.in_actor then
        let {mv_name; mv_type; mv_init} = mvdecl in
          (match mv_type with
              Unit_t ->
                raise (Failure ("Cannot declare value of type unit to " ^ mv_name))
            | Func_t (_, _) ->
                raise (Failure ("Cannot declare mutable funcs types " ^ mv_name))
            | Message_t _ ->
                raise (Failure ("Cannot declare mutable message " ^ mv_name))
            | Actor_t _ | Pool_t _ ->
                raise (Failure ("Cannot spawn mutable actor/pool " ^ mv_name))
            | _ ->
                let texp = check_expr mv_init env in
                let (se, t) = texp in
                let se_t = (match se with
                    SNoexpr -> mv_type
                  | _ -> t
                ) in
                if types_equal se_t mv_type then
                  (if name_taken mv_name env then
                    raise (Failure ("Mutable var " ^ mv_name ^ " declared already"))
                  else
                    let smvdecl = {
                      smv_name = mv_name;
                      smv_type = mv_type;
                      smv_init = texp;
                    } in
                    let nm_table = upd_mvtable_vals smvdecl env.env_mvtable in
                      (SMutdecl smvdecl, upd_env_mvtable nm_table env))
                else
                  raise (Failure ("Variable initialization type mismatch: " ^
                    mv_name ^ " is " ^ (str_types mv_type) ^
                      " but initialized as " ^ (str_types t))))
      else
        raise (Failure ("Mutables types are only allowed in actors"))

    and check_stmt_list (sl : stmt list) (ret : bool) (env : scope) =
      let rec check_f_return (sll : stmt list) =
        try
          let _ = List.find (fun s -> match s with
              Return _ -> true
            | If(_, Block(b1), Block(b2)) ->
                check_f_return b1 || check_f_return b2
            | _ -> false) sll in true
        with Not_found -> false in

      let _ = (
        if ret && not (check_f_return sl) then
          raise (Failure ("This function must return " ^
            (match env.return_t with
                Some t -> str_types t
              | None -> ""
            )))
        else ()
      ) in let (csl, nenv) = (List.fold_left (fun acc st ->
        let (sl', env') = (check_stmt st (snd acc)) in (sl' :: fst acc, env')
      ) ([], env) sl) in (List.rev csl, nenv)

    let check_message_decl (mdecl : message) (env : scope) =
      (try
        let _ = List.find (fun sm -> sm.sm_name = mdecl.m_name) env.messages in
          raise (Failure ("Message " ^ mdecl.m_name ^ " declared already"))
      with Not_found ->
        let smdecl = {
          sm_name = mdecl.m_name;
          sm_formals = mdecl.m_formals
        } in (smdecl, upd_env_messages smdecl env))

    let check_actor_decl (adecl : actor) (env : scope) =
      let check_receive (pat : pattern) (smsgs : smessage list) =
        (try
          let sm = List.find (fun m ->
            let p_formal_ts = get_list_snd (pat.p_mformals) in
            let m_formal_ts = get_list_snd m.sm_formals in
            (pat.p_mid = m.sm_name) && check_args_t m_formal_ts p_formal_ts
          ) smsgs in (Some sm)
        with Not_found -> None) in

      let check_patterns (receive : pattern list) (senv : scope) =
        List.map (fun p ->
          let pvals = spread_arg p.p_mformals [] in
          let pv_table = { vparent = Some senv.env_vtable; svals = pvals } in
          let penv = { senv with
            env_vtable = pv_table;
            actor_init = false;
          } in {
            sp_smid = p.p_mid;
            sp_smformals = p.p_mformals;
            sp_body = fst (check_stmt p.p_body penv)
          }
        ) receive in

      let { a_name; a_formals; a_body; a_receive } = adecl in
        (try
          let _ = List.find (fun ascope ->
            ascope.a_actor.sa_name = a_name
          ) env.actors in
          raise (Failure ("Actor " ^ adecl.a_name ^ " declared already"))
        with Not_found ->
          let rec check_dup l =
            (match l with
                [] -> false
              | (h :: t) ->
                  let x = (List.filter (fun x -> x = h) t) in
                    if (x = []) then
                      check_dup t
                    else
                      true
            ) in
          if check_dup (List.map(fun p -> p.p_mid) a_receive) then
            raise (Failure ("Duplicate pattern matching in receive in actor " ^
              a_name))
          else
            let m_allowed = (List.fold_left
              (fun acc p -> match (check_receive p env.messages) with
                  Some m  -> m::acc
                | None    ->
                    raise (Failure ("Actor " ^ a_name ^
                      " attempts to receive an undefined message " ^ p.p_mid))
              ) [] a_receive
            ) in

            let _ =
              try List.find (fun p -> p.p_mid = "die"
                && List.length p.p_mformals = 0) a_receive
              with Not_found ->
                raise (Failure ("No pattern match for die in actor " ^ a_name))
              in



            let nsvals =
              spread_arg a_formals ([]) in
            let nv_table = { vparent = Some env.env_vtable; svals = nsvals } in
            let curr_scope = { env with
              env_vtable  = nv_table;
              in_actor    = true;
              actor_init  = true
            } in
            let (checked_body, sa_env) = check_stmt a_body curr_scope in
            let sareceive = check_patterns a_receive sa_env in
            let new_sactor = {
              sa_name    = a_name;
              sa_formals = a_formals;
              sa_body    = checked_body;
              sa_receive = sareceive
            } in
            let new_actor_scope = {
              a_actor = new_sactor;
              a_scope = sa_env;
              a_messages = m_allowed
            } in
            let nenv = { env with actors = new_actor_scope :: env.actors } in
              (new_actor_scope, nenv))

    let check_program (p : program) (slib : program) =
      let (p_messages, p_actors, p_functions) = p
      and (sl_messages, sl_actors, sl_functions) = slib in

      let messages = sl_messages @ p_messages
```

```
and actors = sl_actors @ p_actors
and functions = sl_functions @ p_functions in
let sender_ref =  {
  sv_name = "sender";
  sv_type = Unit_t;
  sv_init = (SNoexpr, Unit_t);
} in
let die =
  let die_func = (SFunc_Lit({ sf_formals = []; sf_return_t = Unit_t;
    sf_body = empty_fbody }), Func_t([], Unit_t)) in
  { sv_name = "die"; sv_type = Func_t([], Unit_t); sv_init = die_func } in
let empty_vsymtab = { vparent = None; svals = sender_ref :: [] } in
let empty_mvsymtab = { mvparent = None; smvars =  [] } in
let seed_env = {
  messages = [];
  actors = [];
  env_vtable = upd_vtable_vals die empty_vsymtab;
  env_mvtable = empty_mvsymtab;
  return_t = None;
  in_actor = false;
  actor_init = false;
} in
let (smessages, m_env) = List.fold_left (fun acc m ->
  let (smessage, nenv) = check_message_decl m (snd acc) in
  (smessage :: fst acc, nenv)
) ([], seed_env) messages in
let (sactors, a_env) = List.fold_left (fun acc a ->
  let (a_scope, nenv) = check_actor_decl a (snd acc) in
    (a_scope :: fst acc, nenv)
) ([], m_env) actors in
let (sfunctions, _) = List.fold_left (fun acc f ->
  let (sfunc, nenv) = check_vdecl f (snd acc) in
    match sfunc with
        SVdecl sf -> (sf :: fst acc, nenv)
      | _ -> raise (Failure ("Not a valid function: " ^ f.v_name))
) ([], a_env) functions in
```

```
let main_cnt = List.fold_left (fun acc sf -> if sf.sv_name = "main"
                                then acc + 1 else acc) 0 sfunctions in
match main_cnt with
    0 -> raise (Failure "No main function found in this program")
  | 1 -> (List.rev smessages,
          List.rev sactors,
          List.rev (List.filter (fun sf -> not (sf.sv_name =
                "main")) sfunctions),
          (List.find (fun sf -> sf.sv_name = "main") sfunctions).sv_init)
  | n -> raise (Failure (string_of_int n ^ " main functions found") )
```

8.1.6 Codegen.ml

```
open Ast
open Sast

open Hashtbl

let a_decls = Hashtbl.create 50
let p_decls = Hashtbl.create 50

let std = "std::"
let immut = "immut::"
let mes = "m_"

let actor_include   = "#include <oscar/actor.hpp>\n"
let message_include = "#include <oscar/message.hpp>\n"
let immut_include   = "#include <oscar/immut.hpp>\n"

let rec c_type (t : types) =
  match t with
      Int_t | Bool_t | Double_t | Char_t -> str_types t
    | String_t           -> std ^ str_types t
    | Unit_t             -> "void"
    | Func_t (args, rt)  -> std ^ "function<" ^ c_type rt ^ "(" ^
                             (String.concat ", " (List.map
                               (fun arg -> c_type arg) args)) ^ ")>"
    | List_t t           -> immut ^ "list<" ^ c_cont_type t
    | Set_t t            -> immut ^ "set<" ^ c_cont_type t
    | Map_t (t1, t2)     -> immut ^ "map<" ^ c_type t1 ^ ", " ^ c_cont_type t2
    | Actor_t t          -> "a_" ^ t
    | Pool_t t           -> "pool<" ^ t ^ " *>"
    | Message_t t        -> mes ^ t

and c_cont_type types =
  match types with
```

```
      List_t _ | Set_t _ | Map_t (_, _) -> c_type types ^ " >"
    | _ -> c_type types ^ ">"

and init_list (tel : t_expr list) a =
    String.concat ", " (List.map (fun elem ->
        c_texpr elem a
    ) tel)

and init_tup_list (tel : (t_expr * t_expr) list) =
  let c_tup (sk, sv) = str_texpr sk ^ " , " ^ str_texpr sv in
    String.concat "," (List.map (fun elem ->
      "{ " ^ c_tup elem ^ " }"
    ) tel)

and c_texpr tex (a : bool) =
  let (se, _) = tex in
    (match se with
        SInt_Lit _ | SDouble_Lit _ | SChar_Lit _
      | SBool_Lit _ | SId _ | SNoexpr -> str_texpr tex
      | SUnit_Lit _                -> "void"
      | SString_Lit s              -> std ^ "string(\"" ^ s ^ "\")"
      | SAccess (scont, sit)       -> c_texpr scont a ^ "[" ^ c_texpr sit a ^ "]"
      | SFuncCall (se, sel)        -> se ^ "(" ^ init_list sel a ^ ")"
      | SUop (o, se)               -> str_uop o ^ c_texpr se a
      | SBinop (se1, o, se2)       -> "(" ^ c_texpr se1 a ^ " " ^
                                        str_binop o ^ " " ^ c_texpr se2 a ^ ")"
      | SFunc_Lit sfl              -> c_lambda sfl a
      | SList_Lit (t, sel)         -> immut ^ "list<" ^ str_cont_t t ^ "{" ^
                                        init_list sel a ^ "}"
      | SSet_Lit (t, sel)          -> immut ^ "set<" ^ (str_cont_t t) ^ "{" ^
                                        init_list sel a ^ "}"
      | SMap_Lit (kt, vt, skvs)    -> immut ^ "map<" ^ c_type kt ^ ", " ^
                                        str_cont_t vt ^ "{" ^
                                          init_tup_list skvs ^ "}"
      | SActor_Lit (sat, sel)      -> "new a_" ^ sat ^ "(" ^
                                        init_list sel a ^ ")"
      | SPool_Lit (sat, sel, num)  -> "new pool<a_" ^ sat ^ ">({" ^
                                       init_list sel a ^ "}, " ^
                                        c_texpr num a ^ ")"
      | SMessage_Lit (m, sel)      ->  "new " ^ mes ^ m ^ "(" ^
                                        (init_list sel a) ^
                                          (if List.length sel = 0 then
                                            "" else ", ") ^ (if a then "this"
                                        else "NULL") ^ ")")

and c_sstmt sstmt (a : bool) =
  match sstmt with
      SBlock _              -> "{\n" ^ c_sstmts sstmt a ^ "\n}"
    | SExpr texp            -> c_texpr texp a ^ ";"
    | SReturn se            -> "return " ^ (match (snd se) with
                                   Unit_t -> ""
                                 | _      -> c_texpr se a) ^ ";"
    | SMutdecl smv          -> c_type smv.smv_type ^ " " ^
                                 smv.smv_name ^ (match smv.smv_init with
                                     SNoexpr, _ -> ""
                                   | _ -> "=" ^ (c_texpr smv.smv_init a)) ^ ";"

    | SVdecl sv             -> if a then
                                 c_func sv
                               else
                                 let () = (match sv.sv_type with
                                   Actor_t _ ->
                                     Hashtbl.replace a_decls sv.sv_name true
                                 | Pool_t _ ->
                                     Hashtbl.replace p_decls sv.sv_name true
                                 | _ ->  ()
                                 ) in
                                 "auto " ^ sv.sv_name ^ "=" ^
                                   c_texpr sv.sv_init a ^ ";"
    | SIf (se, s1, s2)      -> c_if se s1 s2 a
    | SActor_send (se, act)  ->
        let a_name = (match fst act with
            SId s -> s
          | _ -> raise (Failure ("sent to not ID (should not happen)"))
        ) in
        let () = (match snd se with
            Message_t "die" -> Hashtbl.remove a_decls a_name
          | _ -> ()
        ) in
        (match a_name with
            "sender"  ->
              "___msgRcvd->sender->receive(" ^ c_texpr se a ^ ");\n"
          | _              ->
              c_texpr act a ^ "->receive(" ^ c_texpr se a ^ ");\n")
    | SPool_send (se, p)  ->
        let p_name = (match fst p with
            SId s -> s
          | _ -> raise (Failure ("sent to not ID (should not happen)"))
        ) in
        let () = (match snd se with
            Message_t "die" -> Hashtbl.remove p_decls p_name
          | _ -> ()
        ) in
        c_texpr p a ^ "->broadcast(" ^ c_texpr se a ^ ");\n"

and c_sstmts sstmts a =
  (match sstmts with
    SBlock sstmts   -> String.concat "\n" (List.map (fun s ->
      c_sstmt s a) sstmts)
  | _           -> "")

and c_formal f = c_type (snd f) ^ " " ^ (match snd f with
    Actor_t _ -> "*"
  | _ -> "") ^ fst f

and c_formals fs = String.concat "," (List.map c_formal fs)

and c_lambda sfl a =
  let { sf_formals = sformals; sf_return_t = srt; sf_body = sbody } = sfl in
    "[&](" ^ c_formals sformals ^ ")" ^ (match srt with
        Unit_t -> ""
      | _ -> "->" ^ c_type srt) ^ c_sstmt sbody a;

and c_func vd =
  let sv_name = vd.sv_name and sv_init = vd.sv_init in
    match fst sv_init with
        SFunc_Lit sfl ->
          let { sf_formals; sf_return_t; sf_body } = sfl in
            c_type sf_return_t ^ " " ^ sv_name ^ "(" ^ c_formals sf_formals ^
              ")\n" ^ c_sstmt sf_body false ^ "\n"
      | _ ->
          raise (Failure ("Top level function declaration failed: " ^
            sv_name ^ " is not a function"))

and c_if te s1 s2 a =
  "if (" ^ (
    let cond = c_texpr te a in
    let cond_len = String.length cond in
      if (String.get cond 0 = '(') &&
        (String.get cond (cond_len - 1) = ')') then
        String.sub cond 1 (cond_len - 2)
      else
        cond
      ) ^ ")\n" ^ c_sstmt s1 a ^ (match s2 with
          SExpr (SNoexpr, _)  -> "\n"
        | _  -> " else " ^ c_sstmt s2 a ^ "\n"
      )

let declare_fields formals =
  String.concat ";\n" (List.map c_formal formals) ^
    if List.length formals > 0 then ";\n" else ""

let constructor_assignment formals =
  String.concat ";\n" (List.map (fun sf ->
    "this->" ^ fst sf ^ " = " ^ fst sf
  ) formals) ^ if List.length formals > 0 then ";\n" else ""

let get_mbody formals =
  "tuple<" ^ String.concat ", " (List.map (fun sf ->
    c_type (snd sf) ) formals) ^ "> get() {\nreturn make_tuple(" ^
      (String.concat ", " (List.map (fun sf -> "this->" ^ fst sf ) formals)) ^
        ");\n}"

let c_message smessage =
  let { sm_name; sm_formals} = smessage in
    "class m_" ^ sm_name ^ " : public Message {\nprivate:\n" ^
      declare_fields sm_formals ^ "\npublic:\nm_" ^ sm_name ^ " (" ^
        c_formals sm_formals ^ (if List.length sm_formals = 0 then "" else ",") ^
          "Actor *sender) : Message(\"" ^ sm_name ^ "\", sender)\n{\n" ^
            constructor_assignment sm_formals ^ "\n}\n" ^
              get_mbody sm_formals ^ "\n};\n"

let declare_queues formals =
  String.concat ";\n" (List.map (fun m ->
    "std::queue<m_" ^ m.sm_name ^ " *> " ^ m.sm_name ^ "Queue"
  ) formals) ^ if List.length formals > 0 then ";\n" else ""

let cast_message message =
  let { sm_name = sm_name ; sm_formals = _} = message in
    "if (m_" ^ sm_name ^ " *pm = dynamic_cast<m_" ^ sm_name ^ " *>(msg)) {\n" ^
      "unique_lock<mutex> lck(mx);\n" ^ sm_name ^ "Queue.push(pm);\n" ^
        "goto notify;\n}\n"

let unpack_body body = String.sub body 1 (String.length body - 2)

(* we give this ___msgRcvd a name so long to avoid conflict *)
let c_pattern sp =
  let { sp_smid = sp_smid; sp_smformals = sp_smformals; sp_body = _ } = sp in
    "void respond(m_" ^ sp_smid ^ " *___msgRcvd) {\n"
    ^ String.concat ";\n"
```

```
      (List.mapi (fun i f ->
        "auto " ^ fst f ^ " = get<" ^ string_of_int i ^
        ">(___msgRcvd->get())"
      ) sp_smformals) ^ (if List.length sp_smformals > 0 then ";\n" else "") ^ (
        let actor_body = c_sstmt sp.sp_body true in
          unpack_body actor_body) ^ "delete ___msgRcvd;\n" ^
      (if sp_smid = "die" then "Die();\n" else "") ^ "}\n"

let consume messages =
    "void consume() {\n" ^
    "unique_lock<mutex> lck(mx);\nwhile (!this->tFinished) {\n" ^
    "while (" ^ (String.concat "&&" (List.map (fun m ->
           m.sm_name ^ "Queue.empty()"
         ) messages)) ^ "){\n" ^
    "if (tFinished)\n" ^
    "return;\n" ^
    "cv.wait(lck);\n" ^
    "}\n" ^
        (String.concat "\n" (List.map (fun m ->
          "if (!" ^ m.sm_name ^ "Queue.empty()) {\n" ^
            "auto msg = " ^ m.sm_name ^ "Queue.front();\n" ^
              m.sm_name ^ "Queue.pop();\n" ^
                "respond(msg);\n" ^
                "goto loop;\n" ^
                "}"
        ) messages)) ^ "\n" ^
    "loop: ;\n" ^
    "}\n" ^
    "}"

let make_die _ =
  "\nvoid Die() {\nthis->tFinished = true;\n" ^
    "___monitor->receive(new DeleteMessage());\ncv.notify_one();\n}\n"

let c_actor sactor_scope =
  let sactor = sactor_scope.a_actor in
  let smessages = sactor_scope.a_messages in
  let { sa_name; sa_formals; sa_body; sa_receive } = sactor in
    "class a_" ^ sactor.sa_name ^ " : public Actor {\nprivate:\n" ^
      declare_fields sa_formals ^ declare_queues smessages ^
        unpack_body (c_sstmt sa_body true) ^ "\n\n" ^
        "public:\n" ^ "a_" ^ sa_name ^
          " (" ^ (c_formals sa_formals) ^ ")" ^
        ": Actor()\n" ^
        "{\n" ^
            constructor_assignment sa_formals ^ "\n" ^
              "___monitor->receive(new SpawnMessage());\n" ^
              "t = thread([=] { consume(); });\n" ^
              "}\nvirtual ~a_" ^ sa_name ^ "() { t.join(); }\n" ^
              (make_die ()) ^
              "virtual void receive(Message *msg) {\n" ^
              String.concat "" (List.map cast_message smessages) ^
              "notify:\n" ^
              "cv.notify_one();\n" ^
              "}\n" ^
              String.concat "\n"
              (List.map c_pattern sa_receive) ^ consume smessages ^ "};\n"

let die_pools_actors s=
  let f_kill k _ acc = "delete " ^ k ^";\n" ^ acc in
  let die_pools = Hashtbl.fold f_kill p_decls s in
  Hashtbl.fold f_kill a_decls die_pools

let main_decl (se, _) =
  match se with
      SFunc_Lit sfl ->
        let sf_formals = sfl.sf_formals and sf_body = sfl.sf_body in
          "int main (" ^ c_formals sf_formals ^
            ") {\n" ^ (
              let sfbody = c_sstmt sf_body false in
              "___monitor = new Monitor();\n" ^ unpack_body sfbody ^
                "\nusleep(1000);" ^
                "\nwhile (!___monitor->is_exitable()) {cout << \"\";}" ^
                  "\ndelete ___monitor;\nreturn 0;\n}"
            ) ^ "\n"
    | _ -> raise (Failure "Main method not found")

let c_program (smessages, sactors, sfuncs, main) =
  actor_include ^ immut_include ^ "static Monitor *___monitor;\n\n" ^
  String.concat "\n" (List.map c_message smessages) ^ "\n\n" ^
  String.concat "\n" (List.map c_actor sactors) ^ "\n" ^
  String.concat "\n\n" (List.map c_func sfuncs) ^ "\n" ^
    main_decl main ^ "\n"
```

8.1.7: optimizer.ml

```
open Ast
open Sast

open Hashtbl

type vscope = {
  loc       : string list;
  block_cnt : int;
  if_cnt    : int;
  func_cnt  : int
}

let vals = Hashtbl.create 200

let mvars = Hashtbl.create 200

(* helpers for optional stuff *)
let option_is_some o =
  match o with
      Some _  -> true
    | None    -> false

let option_is_none o =
  match o with
      Some _  -> false
    | None    -> true

let option_get o =
  match o with
      Some v  -> v
    | None    -> raise (Failure "tried to get None value")

(* helpers for variable tracking *)
```

```
let rec find_val tbl loc s =
  match List.length loc with
      0 -> None
    | _ -> let name = (String.concat "_" loc) ^ "__" ^ s in
            try Some (Hashtbl.find tbl name) with
            | Not_found -> find_val tbl (List.tl loc) s

let build_name scope s =
  (String.concat "_" scope.loc) ^ "__" ^ s

let is_replaceable scope id =
  let match_opt = find_val vals scope.loc id in
  (match match_opt with
      Some (_, rep, _) -> rep
    | None -> false
  )

let add_mvar scope decl =
  let name = build_name scope decl.smv_name in
  Hashtbl.add mvars name ((SNoexpr, Unit_t), false, 0)

let add_val scope decl =
  let {sv_name = id ; sv_type = _ ; sv_init = init} = decl in
  let name = build_name scope id in
  let (sinit, rep) =
    (match (fst init) with
        SInt_Lit _  | SDouble_Lit _ | SChar_Lit _ | SString_Lit _
      | SBool_Lit _ |  SUnit_Lit _  | SId _       -> (init, true)
      (* save space on things we won't fill in later *)
      | _ -> ((SNoexpr, Unit_t), false)
    ) in
  Hashtbl.add vals name (sinit, rep, 0)

let incr_cnt scope id =
  let name = build_name scope id in
  let val_tup = find_val vals scope.loc id in
  match val_tup with
      Some (i, r, c) -> Hashtbl.replace vals name (i, r, c + 1)
    | None ->
        let mvar_tup = find_val mvars scope.loc id in
        let (init, rep, cnt) =
          (match mvar_tup with
              Some (i, r, c) -> (i, r, c)
            | None -> ((SNoexpr, Unit_t), false, 0)
          ) in
        Hashtbl.replace mvars name (init, rep, cnt + 1)

let get_init scope id =
  match (find_val vals scope.loc id) with
      Some (i, _, _) -> i
    | None -> raise (Failure ("no init for name " ^ (build_name scope id)))

let get_cnt scope id =
  let val_tup = find_val vals scope.loc id in
  match val_tup with
      Some (_, _, c) -> c
    | None ->
        let mvar_tup = find_val mvars scope.loc id in
        match mvar_tup with
            Some (_, _, c) -> c
          | None ->
              raise (Failure ("no count for name " ^ (build_name scope id)))

(* optimize a single expression
 * IDs get replaced if they are just a simple value
 * funcs get optimized
 * all actuals used to call anything are optimized
 * uops / binops get reduced if possible *)
let rec opt_expr scope (te : t_expr) =
  let (e, t) = te in
  let ne = (match e with
      SId id ->
        let () = incr_cnt scope id in
        if is_replaceable scope id then
          fst (get_init scope id)
        else
          e
    | SAccess (e1, e2) -> SAccess((opt_expr scope e1), (opt_expr scope e2))
    | SFunc_Lit sf -> opt_func_lit scope sf
    | SList_Lit (typ, exprs) ->
        let nexprs = List.map (opt_expr scope) exprs in
        SList_Lit(typ, nexprs)
    | SSet_Lit (typ, exprs) ->
        let nexprs = List.map (opt_expr scope) exprs in
        SSet_Lit(typ, nexprs)
    | SMap_Lit (t1, t2, kvs) ->
        let nkvs = List.map (fun (k, v) ->
            (opt_expr scope k, opt_expr scope v)) kvs in
        SMap_Lit(t1, t2, nkvs)
    | SMessage_Lit (id, exprs) ->
        let nexprs = List.map (opt_expr scope) exprs in
        SMessage_Lit (id, nexprs)
    | SActor_Lit (id, exprs)->
        let nexprs = List.map (opt_expr scope) exprs in
        SActor_Lit (id, nexprs)
    | SPool_Lit (id, exprs, cnt) ->
        let nexprs = List.map (opt_expr scope) exprs in
        let ncnt = opt_expr scope cnt in
        SPool_Lit (id, nexprs, ncnt)
    | SBinop (e1, op, e2) -> opt_binop scope e1 op e2
    | SUop (uop, e1) -> opt_uop scope uop e1
    | SFuncCall (id, exprs) ->
        let () = incr_cnt scope id in
        let nexprs = List.map (opt_expr scope) exprs in
        SFuncCall (id, nexprs)
    | _ -> e
  ) in
```

```
  (ne, t)

(* optimize the body of the function *)
and opt_func_lit scope (f : sfunc) =
  let fst_body = (opt_outer_block opt_stmt) scope f.sf_body in
  let snd_body = (opt_outer_block snd_stmt) scope fst_body in
  SFunc_Lit({ f with sf_body = snd_body })

(* reduce uops of expressions that can be reduced to literals *)
and opt_uop scope (uop: u_op) (e : t_expr) =
  let (e', t) = opt_expr scope e in
  match uop with
      Not ->
        (match e' with
            SBool_Lit b   -> SBool_Lit (not b)
          | _ -> SUop(uop, (e',t))
        )
    | Neg ->
        (match e' with
            SInt_Lit i    -> SInt_Lit (-i)
          | SDouble_Lit d -> SDouble_Lit (-.d)
          | _ -> SUop(uop, (e',t))
        )

(* reduce binops of expressions that can be reduced to literals *)
and opt_binop scope (e1 : t_expr) (op : bin_op) (e2 : t_expr) =
  let (e1', t1) = opt_expr scope e1 in
  let (e2', t2) = opt_expr scope e2 in
  match op with
      Add ->
        (match e1', e2' with
            SInt_Lit i1, SInt_Lit i2        -> SInt_Lit (i1 + i2)
          | SDouble_Lit d1, SDouble_Lit d2  -> SDouble_Lit (d1 +. d2)
          | SString_Lit s1, SString_Lit s2  -> SString_Lit (s1 ^ s2)
          | _ -> SBinop((e1',t1), op, (e2',t2))
        )
```

```
| Sub   ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 - i2)
      | SDouble_Lit d1, SDouble_Lit d2  -> SDouble_Lit(d1 -. d2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Mult  ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 * i2)
      | SDouble_Lit d1, SDouble_Lit d2  -> SDouble_Lit(d1 *. d2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Div   ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 / i2)
      | SDouble_Lit d1, SDouble_Lit d2  -> SDouble_Lit(d1 /. d2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Less  ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SBool_Lit (i1 < i2)
      | SDouble_Lit d1, SDouble_Lit d2  ->
          SBool_Lit ((compare d1 d2) < 0)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Leq ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SBool_Lit (i1 <= i2)
      | SDouble_Lit d1, SDouble_Lit d2  ->
          SBool_Lit ((compare d1 d2) <= 0)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Greater ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SBool_Lit (i1 > i2)
      | SDouble_Lit d1, SDouble_Lit d2  ->
```

```
          SBool_Lit ((compare d1 d2) > 0)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Geq ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SBool_Lit (i1 >= i2)
      | SDouble_Lit d1, SDouble_Lit d2  ->
          SBool_Lit ((compare d1 d2) >= 0)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Mod ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 mod i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Bit_And ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 land i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Bit_Or ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 lor i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Bit_Xor ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 lxor i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Bit_RShift ->
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 lsr i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Bit_LShift ->
```

```
    (match e1', e2' with
        SInt_Lit i1, SInt_Lit i2      -> SInt_Lit (i1 lsl i2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Equal ->
    (match e1', e2' with
        SInt_Lit v1, SInt_Lit v2            -> SBool_Lit (v1 = v2)
      | SDouble_Lit v1, SDouble_Lit v2      ->
          SBool_Lit ((compare v1 v2) = 0)
      | SChar_Lit v1, SChar_Lit v2          -> SBool_Lit (v1 = v2)
      | SString_Lit v1, SString_Lit v2      -> SBool_Lit (v1 = v2)
      | SBool_Lit v1, SBool_Lit v2          -> SBool_Lit (v1 = v2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Neq ->
    (match e1', e2' with
        SInt_Lit v1, SInt_Lit v2            -> SBool_Lit (v1 != v2)
      | SDouble_Lit v1, SDouble_Lit v2      ->
          SBool_Lit ((compare v1 v2) != 0)
      | SChar_Lit v1, SChar_Lit v2          -> SBool_Lit (v1 != v2)
      | SString_Lit v1, SString_Lit v2      -> SBool_Lit (v1 != v2)
      | SBool_Lit v1, SBool_Lit v2          -> SBool_Lit (v1 != v2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| And ->
    (match e1', e2' with
        SBool_Lit b1, SBool_Lit b2    -> SBool_Lit(b1 && b2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Or ->
    (match e1', e2' with
        SBool_Lit b1, SBool_Lit b2    -> SBool_Lit(b1 || b2)
      | _ -> SBinop((e1', t1), op, (e2', t2))
    )
| Assign ->
    SBinop((e1', t1), op, (e2', t2))
```

```
(* optimize a statement, returning Some sstmt if useful else None
 * cleans up all statements and expressions
 * ignores variable declaration if the variable init value can be
 * reduced to a literal or an id of another variable *)
and opt_stmt scope (s : sstmt) =
  match s with
      SBlock stmts -> opt_block opt_stmt scope stmts
    | SExpr (e, t) ->
        (match e with
            SFuncCall _ ->
              let nexpr = opt_expr scope (e, t) in
              (Some (SExpr(nexpr)), scope)
          | SBinop (_, Assign, _) ->
              let nexpr = opt_expr scope (e, t) in
              (Some (SExpr(nexpr)), scope)
          | _ -> (None, scope)
        )
    | SReturn e ->
        let nexpr = opt_expr scope e in
        (Some (SReturn(nexpr)), scope)
    | SVdecl vd -> opt_valdecl scope vd
    | SMutdecl md -> opt_mutdecl scope md
    | SIf (spred, sif, selse) -> opt_if scope spred sif selse
    | SActor_send (e1, e2) ->
        let nmess = opt_expr scope e1 in
        let nact = opt_expr scope e2 in
        (Some (SActor_send(nmess, nact)), scope)
    | SPool_send (e1, e2) ->
        let nmess = opt_expr scope e1 in
        let npool = opt_expr scope e2 in
        (Some (SPool_send(nmess, npool)), scope)

(* handling the "inner" blocks that can be nested *)
and opt_block stmt_func scope stmts =
  let nscope =
    if scope.block_cnt = 0 then
      { scope with block_cnt = 1 }
    else
      let new_name = "b" ^ (string_of_int scope.block_cnt) in
      { loc = new_name :: scope.loc; block_cnt = 0;
          if_cnt = 0; func_cnt = 0 }
    in
  let acc_fun (scope, slist) stmt =
    let (nstmt, nscope) = stmt_func scope stmt in
    if option_is_some nstmt then
      (nscope, (option_get nstmt) :: slist)
    else
      (nscope, slist)
  in
  let (_, nstmts) = List.fold_left acc_fun (nscope, []) stmts in

  let ret_scope = { scope with block_cnt = scope.block_cnt + 1 } in

  (match nstmts with
      []  -> (None, ret_scope)
    | _   -> ((Some (SBlock (List.rev nstmts))), ret_scope)
  )

(* this is to handle "higher-level" blocks that always must exist *)
and opt_outer_block stmt_func scope block =
  let stmts =
    (match block with
      SBlock s -> s
    | _ -> raise (Failure "opt_inner_block called without block")
    ) in
  let o_block = fst (opt_block stmt_func scope stmts) in
  if option_is_some o_block then
    option_get o_block
  else
    SBlock([])
```

```
(* optimize immutable variable declaration
 * if we save the value of the variable, don't actually put the
 * declaration in code *)
and opt_valdecl scope vd =
  let opt_vd = { vd with sv_init = opt_expr scope vd.sv_init } in
  let () = add_val scope opt_vd in
  let nstmt =
    (if (is_replaceable scope vd.sv_name) then
      None
    else
      Some (SVdecl (opt_vd) )
    ) in
  (nstmt, scope)

(* optimize mutable varialbe declaration
 * always ends up in code, but init value is optimize *)
and opt_mutdecl scope md =
  let () = add_mvar scope md in
  let opt_md = { md with smv_init = opt_expr scope md.smv_init } in
  (Some ( SMutdecl (opt_md) ), scope)


(* optimize if/else block
 * if we can reduce the predicate to a boolean literal
 * then replace the if with a block of the used part *)
and opt_if scope pred ifb elseb =
  let clean_scope scope name =
    let new_name =
      (if scope.if_cnt = 0 then
        name
      else
        (name ^ (string_of_int scope.if_cnt))
      ) in
    { loc = new_name :: scope.loc; block_cnt = 0; if_cnt = 0; func_cnt = 0 }
  in
  let (ne, nt) = opt_expr scope pred in

  let nstmt = (match ne with
      SBool_Lit b ->
        (if b then
          fst (opt_stmt (clean_scope scope "if") ifb)
        else
          fst (opt_stmt (clean_scope scope "else") elseb)
        )
    | _ ->
        let nifb_o = fst(opt_stmt (clean_scope scope "if") ifb) in
        let nelseb_o = fst(opt_stmt (clean_scope scope "else") elseb) in
        if option_is_none nifb_o && option_is_none nelseb_o then
          None
        else
          let nifb =
            (if option_is_some nifb_o then
              option_get nifb_o
            else
              SExpr((SNoexpr, Unit_t))
            ) in
          let nelseb =
            (if option_is_some nelseb_o then
              option_get nelseb_o
            else
              SExpr((SNoexpr, Unit_t))
            ) in
          Some (SIf((ne, nt), nifb, nelseb)))
  in
  (nstmt, { scope with if_cnt = scope.if_cnt + 1 })

and snd_stmt scope s =
  match s with
      SBlock stmts -> opt_block snd_stmt scope stmts
    | SVdecl vd -> snd_vdecl scope vd
    | SMutdecl md -> snd_mdecl scope md
    | SIf (spred, sif, selse) -> snd_if scope spred sif selse
    | _ -> (Some s, scope)

and snd_vdecl scope vd =
  if (get_cnt scope vd.sv_name) = 0 then
    (None, scope)
  else
    (Some (SVdecl vd), scope)

and snd_mdecl scope md =
  if (get_cnt scope md.smv_name) = 0 then
    (None, scope)
  else
    (Some (SMutdecl md), scope)

and snd_if scope spred ifb elseb =
  let clean_scope scope name =
    let new_name =
      (if scope.if_cnt = 0 then
        name
      else
        (name ^ (string_of_int scope.if_cnt))
      ) in
    { loc = new_name :: scope.loc; block_cnt = 0; if_cnt = 0; func_cnt = 0 }
  in
  let nifb = opt_outer_block snd_stmt (clean_scope scope "if") ifb in
  let nelseb = opt_outer_block snd_stmt (clean_scope scope "else") elseb in
  (Some (SIf(spred, nifb, nelseb)), { scope with if_cnt = scope.if_cnt + 1 })

let opt_funcdecl scope f =
  let () = add_val scope f in
  let nscope = { loc = f.sv_name :: scope.loc; block_cnt = 0;
                 if_cnt = 0; func_cnt = 0 } in
  { f with sv_init = opt_expr nscope f.sv_init }

let snd_funcdecl scope f =
  if (get_cnt scope f.sv_name) = 0 then
    None
  else
    Some f

(* optimize a pattern: reduce the body *)
let opt_pattern scope spat =
  let nscope = { scope with loc = spat.sp_smid :: scope.loc } in
  let nbody = (opt_outer_block opt_stmt) nscope spat.sp_body in
  { spat with sp_body = nbody }

(* optimize an actor: reduce the body and optimize all patterns *)
let opt_actor scope sact =
  let nscope = { loc = sact.sa_name :: scope.loc;
                 block_cnt = 0; if_cnt = 0; func_cnt = 0 } in

  let nbody = (opt_outer_block opt_stmt) nscope sact.sa_body in
  let nreceive = List.map (opt_pattern nscope) sact.sa_receive in
  { sact with sa_body = nbody; sa_receive = nreceive }

let optimize_program (messages, actor_scopes, functions, main) =
  let main_scope =
    { loc = ["main"]; block_cnt = 0; if_cnt = 0; func_cnt = 0 } in
  let actors = List.map (fun a -> a.a_actor) actor_scopes in
  let fp_actors = List.map (opt_actor main_scope) actors in
  let fp_functions = List.map (opt_funcdecl main_scope) functions in
  let (main_lit, main_typ) = main in
  let fp_main = match (main_lit) with
        SFunc_Lit(f) -> opt_func_lit main_scope f
      | _ -> raise (Failure "bad main") in

  let fp_actor_scopes =
    List.map2 (fun a a_s -> { a_s with a_actor = a }) fp_actors actor_scopes in
  let sp_functions = List.map option_get (List.filter option_is_some
      (List.map (snd_funcdecl main_scope) fp_functions)) in
  (messages, fp_actor_scopes, sp_functions, (fp_main, main_typ)
```

8.2: C++ Backend

8.2.1 actor.hpp

```
#ifndef __ACTOR__
#define __ACTOR__

#include <unistd.h>
#include "iostream"
#include <cstdio>
#include <queue>
#include <string>
#include <unordered_map>

#include <condition_variable>
#include <mutex>
#include <thread>
#include <atomic>

#include "message.hpp"

using namespace std;

class Actor {
public:
    std::thread t;
    condition_variable cv;
    mutex mx;
    bool tFinished;

public:
    Actor() : tFinished(false) { }

    ~Actor() { }
    virtual void receive(Message* const msg) = 0;

    void Die() { this->tFinished = true; }
};

class Monitor : public Actor {
    bool end;
    atomic<int> actor_counter;

    queue<SpawnMessage *> spawnQueue;
    queue<DeleteMessage *> deleteQueue;

    void Die() { this->tFinished = true; this->end = true; }

    void consume() {
        unique_lock<mutex> lck(mx);

        while (!this->tFinished) {
            while (spawnQueue.empty() && deleteQueue.empty()) {
                if (tFinished)
                    return;
                cv.wait(lck);
            }

            if (!spawnQueue.empty()) {
                auto msg = spawnQueue.front();
                spawnQueue.pop();
                respond(msg);
            } else if (!deleteQueue.empty()) {
                auto msg = deleteQueue.front();
                deleteQueue.pop();
                respond(msg);
            }
        }
        this->end = true;
    }
    void respond(SpawnMessage *msg) {
        actor_counter++;
        end = false;

        delete msg;
    }

    void respond(DeleteMessage *msg) {
        actor_counter--;

        delete msg;
        if (actor_counter.load() == 0 && spawnQueue.empty() &&
            deleteQueue.empty())
            Die();
    }

public:
    Monitor() {
        this->end = true;

        actor_counter = 0;
        t = thread([=] { consume(); });
    }

    virtual ~Monitor() {
        cv.notify_one();
        Die();

        t.join();
    }

    bool is_exitable() { return this->end; }

    void receive(Message* const msg) {
        if (SpawnMessage* pm = dynamic_cast<SpawnMessage *>(msg)) {
            unique_lock<mutex> lck(mx);
            spawnQueue.push(pm);
        } else if (DeleteMessage *pm = dynamic_cast<DeleteMessage *>(msg)) {
            unique_lock<mutex> lck(mx);
            deleteQueue.push(pm);
        }

        cv.notify_one();
    }
};

#endif  // __ACTOR__
```

8.2.2 message.hpp

```
#ifndef __MESSAGE__
#define __MESSAGE__

#include <string>
#include <tuple>
#include <vector>

#include "actor.hpp"

using namespace std;

class Actor;

class Message {
// todo: make private
public:
    string name;
    Actor* sender;

public:
    Message(const string& name = "", Actor* const sender = NULL) :
        name(name), sender(sender) {}

    virtual ~Message() {}
};

// Ping messages
class SpawnMessage : public Message {
public:
    SpawnMessage() : Message("spawn") { }
};

// Ping messages
```

```
class DeleteMessage : public Message {
public:
    DeleteMessage() : Message("delete") { }
};

#endif //__MESSAGE__
```

8.2.3 immut.hpp

```
#ifndef __IMMUT_HPP__
#define __IMMUT_HPP__

#include "list.hpp"
#include "map.hpp"
#include "set.hpp"

#include <iomanip>
#include <string>

template <typename T, typename F>
void ForEach(const F f, const immut::list<T> &t)
{
    t.forEach(f);
}

template <typename T, typename F>
void ForEach(const F f, const immut::set<T> &t)
{
    t.forEach(f);
}

template <typename K, typename V, typename F>
void ForEach(const F f, const immut::map<K, V> &t)
{
    t.forEach(f);
}

template<class T, class U, class F>
U FoldLeft(F f, U acc, immut::list<T> lst)
{
    static_assert(is_convertible<F, function<U(U, T)>>::value,
        "FoldLeft requires a function type U(U, T)");
    ForEach([&f, &acc](const T &v){
        acc = f(acc, v);
    }, lst);

    return acc;
}

template<class T>
immut::list<T> Reverse(immut::list<T> const &lst)
{
    return FoldLeft([](immut::list<T> const &acc, T v) {
        return immut::list<T>(v, acc);
    }, immut::list<T>(), lst);
}

int Size(string s)
{
    return s.size();
}

template <typename T>
int Size(immut::list<T> l)
{
    return FoldLeft([](size_t acc, T t) -> size_t {
        return acc + 1;
    }, 0, l);
}

template <typename T>
int Size(immut::set<T> s)
{
    size_t size = 0;

    ForEach([&size](T t) {
        size++;
    }, s);

    return size;
}

template <typename K, typename V>
int Size(immut::map<K, V> m)
{
    size_t size = 0;

    ForEach([&size](K k, V v) {
        size++;
    }, m);

    return size;
}


template <typename T>
immut::set<T> Put(T x, immut::set<T> s)
{
    return s.inserted(x);
}

template <typename K, typename V>
immut::map<K, V> Put(K k, V v, immut::map<K, V> m)
{
    return m.inserted(k, v);
}

template <typename T, typename F>
immut::list<T> Filter(F f, immut::list<T> t)
{
    static_assert(is_convertible<F, function<bool(T)>>::value,
        "Filter requires a function type bool(T)");
    immut::list<T> res;

    ForEach([&f, &res, &t](T const &v){
        if (f(v))
            res = res.push_front(v);
    }, t);

    return Reverse<T>(res);
}

template <typename T, typename F>
immut::set<T> Filter(F f, immut::set<T> t)
{
    static_assert(is_convertible<F, function<bool(T)>>::value,
        "Filter requires a function type bool(T)");

    immut::set<T> res;

    ForEach([&f, &res, &t](T const &v){
        if (f(v))
            res = Put(v, res.inserted(v));
    }, t);

    return res;
}

template <typename K, typename V, typename F>
immut::map<K, V> Filter(F f, immut::map<K, V> t)
{
    static_assert(is_convertible<F, function<bool(K, V)>>::value,
        "Filter requires a function type bool(K, V)");

    immut::map<K, V> res;

    ForEach([&f, &res, &t](const K &k, const V &v){
        if (f(k, v))
```

```
            res = Put(k, v, res);
    }, t);

    return res;
}

template <typename T>
bool Contains(T t, immut::list<T> l)
{
    return l.contains(t);
}

template <typename T>
bool Contains(T t, immut::set<T> s)
{
    return s.contains(t);
}

template <typename K, typename V>
bool Contains(K k, immut::map<K, V> m)
{
    return m.contains(k);
}

template<class T, class F>
auto Map(F f, immut::list<T> t) -> immut::list<decltype(f(t.front()))>
{
    using U = decltype(f(t.front()));

    static_assert(is_convertible<F, function<U(T)>>::value,
        "Map requires a function type U(T)");

    immut::list<U> res;

    t.forEach([&f, &res](const T &v) { res = res.push_front(f(v)); });
    return Reverse<U>(res);
}

template<class T, class F>
auto Map(F f, immut::set<T> t) -> immut::set<decltype(f(t.front()))>
{
    using U = decltype(f(t.front()));

    static_assert(is_convertible<F, function<U(T)>>::value,
        "Met requires a function type U(T)");

    immut::set<U> res;

    t.forEach([&f, &res](const T &v) { res = Put<U>(f(v), res); });

    return res;
}

template <typename K, typename V, typename F>
auto Map(F f, immut::map<K, V> t) ->
    immut::map<K, decltype(f(t.rootKey(), t.rootValue()))>
{
    using U = decltype(f(t.rootKey(), t.rootValue()));

    static_assert(is_convertible<F, function<U(K, V)>>::value,
        "Map requires a function type U(K, V)");

    immut::map<K, U> res;

    t.forEach([&f, &res](K const &k, V const &v) {
        res = Put(k, f(k, v), res);
    });

    return res;
}
```

```
template <typename T, typename U, typename F>
U Reduce(F f, immut::list<T> lst)
{
    static_assert(is_convertible<F, function<U(T, T)>>::value,
        "Reduce requires a function type U(T, T)");
    static_assert(!lst.isEmpty(),
        "Reduce requires a nonempty collection");

    auto next = lst.popped_front;

    return FoldLeft(f, f(lst.front(), next.front()), next.popped_front());
}

template <typename T>
immut::list<T> MergeFront(immut::list<T> const &a, immut::list<T> const &b)
{
    if (a.isEmpty())
        return b;

    return immut::list<T>(a.front(), MergeFront(a.pop_front(), b));
}

template <typename T>
immut::list<T> MergeBack(immut::list<T> const &a, immut::list<T> const &b)
{
    return MergeFront(b, a);
}

template <typename T>
immut::list<T> PopFront(immut::list<T> const l)
{
    return l.pop_front();
}

template <typename T>
immut::list<T> PopBack(immut::list<T> const l)
```

```
{
    size_t size = Size(l) - 1;

    return l.removeAt(0, size);
}

template <typename T>
immut::list<T> Prepend(T t, immut::list<T> const l)
{
    return l.push_front(t);
}

template <typename T>
immut::list<T> Append(T t, immut::list<T> const l)
{
    auto r1 = Reverse(l);
    auto r2 = r1.push_front(t);

    return Reverse(r2);
}

template <typename T>
immut::set<T> Union(immut::set<T> const & a, immut::set<T> const & b)
{
    immut::set<T> res = a;

    ForEach([&res, &a](const T &v){
        if (!a.contains(v))
            res = res.inserted(v);
    }, b);

    return res;
}

template<class T>
immut::set<T> Diff(immut::set<T> const & a, immut::set<T> const & b)
```

```
{
    immut::set<T> res;

    ForEach([&res, &b](const T &v){
        if (!b.contains(v))
            res = res.inserted(v);
    }, a);

    return res;
}

template<class T>
immut::set<T> Intersection(immut::set<T> const & a, immut::set<T> const & b)
{
    immut::set<T> res;

    ForEach([&res, &a](const T &v){
        if (a.contains(v))
            res = res.inserted(v);
    }, b);

    return res;
}

template<class T>
bool SubSet(immut::set<T> const & a, immut::set<T> const & b)
{
    bool is_subset = true;

    ForEach([&is_subset, &b](const T &v) {
        if (!b.contains(v))
            is_subset = false;
    }, a);

    return is_subset;
}
```

```
int AsInt(double d) { return int(d); }

double AsDouble(int i) { return double(i); }

std::string AsString(int i) { return std::to_string(i); }

std::string AsString(char c) { return std::to_string(c); }

std::string AsString(bool b) { return std::to_string(b); }

std::string AsString(double d) { return std::to_string(d); }

template <typename T>
std::string AsString(immut::list<T> l) { return l.str(); }

template <typename T>
std::string AsString(immut::set<T> s) { return s.str(); }

template <typename K, typename V>
std::string AsString(immut::map<K, V> m) { return m.str(); }

void Println(int i) { std::cout << i << std::endl; }

void Println(char c) { std::cout << c << std::endl; }

void Println(bool b) {
    std::cout << (b == true ? "true" : "false") << std::endl;
}

void Println(double d) {
    std::cout << fixed << setprecision(10);
    std::cout << d << std::endl;
}

void Println(string s) { std::cout << s << std::endl; }
```

```
template <typename T>
void Println(immut::list<T> l) { std::cout << l << std::endl; }

template <typename T>
void Println(immut::set<T> s) { std::cout << s << std::endl; }

template <typename K, typename V>
void Println(immut::map<K, V> m) { std::cout << m << std::endl; }

#endif
```

8.2.4 list.hpp

```
#ifndef __LIST_HPP__
#define __LIST_HPP__

#include "collection.hpp"

using namespace std;

namespace immut {
    template <typename T>
    class list : public collection<T> {
        Type type = L;

        struct Item {
            Item(T v, shared_ptr<const Item> tail) :
                _hash(rand()), _val(v), _next(move(tail)) {}

            explicit Item(T v) : _val(v) {}

            int _hash;
            T _val;

            shared_ptr<const Item> _next;
        };

        friend Item;

        shared_ptr<const Item> _head;

        explicit list(shared_ptr<const Item> items) : _head(move(items)) {}

    public:
        list() {}
        list(T v, list const &tail) :
            _head(make_shared<Item>(v, tail._head)) {}

        explicit list(T v) : _head(make_shared<Item>(v)) {}

        list(initializer_list<T> init)
        {
            for (auto it = rbegin(init); it != rend(init); ++it) {
                _head = make_shared<Item>(*it, _head);
            }
        }

        shared_ptr<const Item> get_root() const { return _head; }

        virtual bool isEmpty() const { return !_head; }

        virtual T front() const
        {
            assert(!isEmpty());

            return _head->_val;
        }

        list pop_front() const
        {
            assert(!isEmpty());

            return list(_head->_next);
        }

        list push_front(T v) const { return list(v, *this); }

        list take(int n)
        {
            if (n <= 0 || isEmpty())
              return list();
            return pop_front().take(n - 1).push_front(front());
        }

        list insertedAt(int i, T v) const
        {
            if (i == 0)
                return push_front(v);
            else {
                assert(!isEmpty());

                return list<T>(front(), pop_front().insertedAt(i - 1, v));
            }
        }

        list removeAt(int i, int j) const
        {
            if (isEmpty())
              return list();

            if (i == j)
                return pop_front();

            return list(front(), pop_front().removeAt(i + 1, j));
        }

        list removed(T v) const
        {
            if (isEmpty())
              return list();

            if (v == front())
                return pop_front();

            return list(front(), pop_front().removed(v));
        }
```

```
bool contains(T v) const
{
    if (isEmpty())
      return false;

    if (v == front())
      return true;

    return pop_front().contains(v);
};

template <typename F>
void forEach(const F f) const
{
    Item const *it = get_root().get();

    while (it != nullptr) {
        f(it->_val);

        it = it->_next.get();
    }
}

bool operator==(const list &rhs)
{
    Item const *it1 = get_root().get();
    Item const *it2 = rhs.get_root().get();

    while (!(it1 == nullptr || it2 == nullptr)) {
        if (it1->_val != it2->_val)
            return false;

        it1 = it1->_next.get();
        it2 = it2->_next.get();
    }
    if ((it1 == nullptr && it2 != nullptr) ||
          (it1 != nullptr && it2 == nullptr))
      return false;

    return true;
}

bool operator > (const list &rhs)
{
    return get_root().get()->_hash > rhs.get_root().get()->_hash;
}

bool operator < (const list &rhs)
{
    return get_root().get()->_hash < rhs.get_root().get()->_hash;
}

const T operator[](int idx) const
{
    const Item *it1 = get_root().get();

    while (idx--) {
        if (it1 == nullptr)
            throw invalid_argument("Index out of bound");

        it1 = it1->_next.get();
    }

    return it1->_val;
}

string str() const
{
    stringstream os;
```

```
            os << "[ ";

            forEach([&os](T v) {
                os << v << " ";
            });

            os << "]";

            return os.str();
        }

        friend ostream &operator<<(ostream &os, const list &t)
        {
            return os << t.str();
        }
    };
}

#endif
```

8.2.5 map.hpp

```
#ifndef __MAP_HPP__
#define __MAP_HPP__

#include "collection.hpp"

using namespace std;

namespace immut {
    template <typename K, typename V>
    class map : public collection<K>
    {
        Type type = M;

        struct KVNode
        {
            KVNode(Color c, shared_ptr<const KVNode> const &lft,
                K key, V val, shared_ptr<const KVNode> const & rgt)
            {
                _hash = rand();
                _c = c;
                _lft = lft;
                _rgt = rgt;
                _key = key;
                _val = val;
            }

            int _hash;

            Color _c;
            K _key;
            V _val;
            shared_ptr<const KVNode> _lft;
            shared_ptr<const KVNode> _rgt;
        };

        explicit map(shared_ptr<const KVNode> const &node) {
            _root = node;
        }

        shared_ptr<const KVNode> _root;

        Color rootColor() const
        {
            assert(!isEmpty());

            return _root->_c;
        }

        map paint(Color c) const
        {
            assert(!this->isEmpty());

            return map(c, left(), this->rootKey(), this->rootValue(), right());
        }

        int countB() const
        {
            if (this->isEmpty())
                return 0;

            int lft = left().countB();

            assert(lft == right().countB());

            return (this->rootColor() == B)? 1 + lft: lft;
        }
```

```
map left() const
{
    assert(!this->isEmpty());

    return map(_root->_lft);
}

map right() const
{
    assert(!this->isEmpty());

    return map(_root->_rgt);
}

bool doubledLeft() const
{
    return !this->isEmpty() && this->rootColor() == R &&
        !this->left().isEmpty() && this->left().rootColor() == R;
}

bool doubledRight() const
{
    return !this->isEmpty() && this->rootColor() == R &&
        !this->right().isEmpty() && this->right().rootColor() == R;
}

map balance(const map &lft, K x, V v, const map &rgt) const
{
    if (lft.doubledLeft()) {
        return map(R, lft.left().paint(B), lft.rootKey(),
            lft.rootValue(), map(B, lft.right(), x, v, rgt));
    } else if (lft.doubledRight()) {
        return map(R, map(B, lft.left(), lft.rootKey(), lft.rootValue(),
            lft.right().left()), lft.right().rootKey(),
            lft.right().rootValue(), map(B, lft.right().right(),
            x, v, rgt));
        } else if (rgt.doubledLeft()) {
            return map(R, map(B, lft, x, v, rgt.left().left()),
                rgt.left().rootKey(), rgt.left().rootValue(),
                map(B, rgt.left().right(), rgt.rootKey(),
                rgt.rootValue(), rgt.right()));
        } else if (rgt.doubledRight()) {
            return map(R, map(B, lft, x, v, rgt.left()), rgt.rootKey(),
                rgt.rootValue(), rgt.right().paint(B));
        } else
            return map(B, lft, x, v, rgt);
    }

    virtual void assert1() const
    {
        if (!this->isEmpty()) {
            auto lft = left();
            auto rgt = right();

            if (this->rootColor() == R) {
                assert(lft.isEmpty() || lft.rootColor() == B);
                assert(rgt.isEmpty() || rgt.rootColor() == B);
            }

            lft.assert1();
            rgt.assert1();
        }
    }

public:
    map() {}

    map(Color c, const map &lft, K key, V val, const map &rgt) :
        _root(make_shared<const KVNode>(c, lft._root, key, val, rgt._root))
    {
        assert(lft.isEmpty() || lft.rootKey() < key);
        assert(rgt.isEmpty() || key < rgt.rootKey());
```

```
}

map(initializer_list<pair<K, V> > initMap)
{
    map<K, V> m;

    for (auto it = initMap.begin(); it != initMap.end(); ++it)
        m = m.inserted(it->first, it->second);

    this->_root = m.get_root();
}

shared_ptr<const KVNode> get_root() { return _root; }

K rootKey() const
{
    assert(!this->isEmpty());

    return _root->_key;
}

V rootValue() const
{
    assert(!this->isEmpty());

    return _root->_val;
}

bool isEmpty() const { return !_root; }

template <typename F>
void forEach(F f)  const
{
    static_assert(is_convertible<F, function<void(K, V)>>::value,
         "ForEach requires a function type void(K, V)");
    if (!this->isEmpty()) {
        this->left().forEach(f);

        f(this->rootKey(), rootValue());

        this->right().forEach(f);
    }
}

map insert(K x, V v) const
{
    this->assert1();

    if (this->isEmpty())
        return map(R, map(), x, v, map());

    K y = this->rootKey();
    V yv = this->rootValue();

    if (x == y)
        return map(B, this->left(), x, yv, this->right());

    Color c = this->rootColor();

    if (this->rootColor() == B) {
        if (x < y) {
            return balance(this->left().insert(x, v), y, yv,
                this->right());
        }

        return balance(this->left(), y, yv, this->right().insert(x, v));
    } else {
        if (x < y) {
            return map(c, this->left().insert(x, v),
                y, yv, this->right());
        }
```

```
            return map(c, this->left(), y, yv, this->right().insert(x, v));
        }
    }

    map  inserted(K x, V v) const
    {
        auto t = insert(x, v);

        return map(B, t.left(), t.rootKey(), t.rootValue(), t.right());
    }

    virtual bool contains(K x) const
    {
        if (this->isEmpty())
            return false;

        K y = this->rootKey();

        if (x < y)
            return left().contains(x);
        else if (y < x)
            return right().contains(x);
        else
            return true;
    }

    V find(K key) const
    {
        if (this->isEmpty())
            throw out_of_range("Key not found");

        K y = this->rootKey();

        if (key < y)
            return left().find(key);
        else if (y < key)
            return right().find(key);
        else
            return rootValue();
    }

    const V operator[](K key) const
    {
        return find(key);
    }

    bool operator==(const map &rhs)
    {
        bool ret = true;

        forEach([&ret, &rhs](K k, V v) {
            if (!rhs.contains(k) || rhs.find(k) != v)
                ret = false;
        });

        return ret;
    }

    bool operator > (const map &rhs)
    {
        return get_root().get()->_hash > rhs.get_root().get()->_hash;
    }

    bool operator < (const map &rhs)
    {
        return get_root().get()->_hash < rhs.get_root().get()->_hash;
    }

    string str() const
    {
        stringstream os;
```

```
            os << "[ ";

            forEach([&os](K k, V v) {
                os << k << " -> " << v << " ";
            });

            os << "]";

            return os.str();
        }

        friend ostream &operator<<(ostream &os, const map &m)
        {
            return os << m.str();
        }
    };
}
#endif
```

8.2.6 set.hpp

```
#ifndef __SET_HPP__
#define __SET_HPP__

#include "collection.hpp"

using namespace std;

namespace immut {
    template <typename T>
    class set : public collection<T> {
        Type type = S;

        struct Node {
            Node(Color c, shared_ptr<const Node> const &lft, T val,
                shared_ptr<const Node> const & rgt) :
                _hash(rand()), _c(c), _val(val), _lft(lft), _rgt(rgt) {}

            int _hash;
            Color _c;
            T _val;

            shared_ptr<const Node> _lft;
            shared_ptr<const Node> _rgt;
        };

        explicit set(const shared_ptr<const Node> &node) : _root(node) {}

        shared_ptr<const Node> _root;

        Color rootColor() const
        {
            assert(!isEmpty());
            return _root->_c;
        }

        set paint(Color c) const
        {
            assert(!this->isEmpty());

            return set(c, left(), this->front(), right());
        }

        set left() const
        {
            assert(!this->isEmpty());

            return set(this->_root->_lft);
        }

        set right() const
        {
            assert(!this->isEmpty());

            return set(this->_root->_rgt);
        }

        bool doubledLeft() const
        {
            return !this->isEmpty() && this->rootColor() == R &&
                !this->left().isEmpty() && this->left().rootColor() == R;
        }

        bool doubledRight() const
        {
            return !this->isEmpty() && this->rootColor() == R &&
                !this->right().isEmpty() && this->right().rootColor() == R;
        }
```

```
set balance(set const & lft, T x, set const & rgt) const
{
    if (lft.doubledLeft()) {
        return set(R, lft.left().paint(B), lft.front(),
            set(B, lft.right(), x, rgt));
    } else if (lft.doubledRight()) {
        return set(R, set(B, lft.left(), lft.front(),
            lft.right().left()), lft.right().front(),
            set(B, lft.right().right(), x, rgt));
    } else if (rgt.doubledLeft()) {
        return set(R, set(B, lft, x, rgt.left().left()),
            rgt.left().front(), set(B, rgt.left().right(),
            rgt.front(), rgt.right()));
    } else if (rgt.doubledRight()) {
        return set(R, set(B, lft, x, rgt.left()),
            rgt.front(), rgt.right().paint(B));
    } else
        return set(B, lft, x, rgt);
}

virtual void assert1() const
{
    if (!this->isEmpty()) {
        auto lft = left();
        auto rgt = right();

        if (this->rootColor() == R) {
            assert(lft.isEmpty() || lft.rootColor() == B);
            assert(rgt.isEmpty() || rgt.rootColor() == B);
        }

        lft.assert1();
        rgt.assert1();
    }
}
```

```
public:
    set() {}

    set(Color c, const set &lft, T val, const set &rgt)
        : _root(make_shared<const Node>(c, lft._root, val, rgt._root))
    {
        assert(lft.isEmpty() || lft.front() < val);
        assert(rgt.isEmpty() || val < rgt.front());
    }

    set<T> (initializer_list<T> init)
    {
        set<T> t;

        for (T v : init)
            t = t.inserted(v);

        this->_root = t.get_root();
    }

    shared_ptr<const Node> get_root() { return _root; }

    T front() const { return _root->_val; }

    bool isEmpty() const { return !_root; }

    template <typename F>
    void forEach(F f) const {
        static_assert(is_convertible<F, function<void(T)>>::value,
             "ForEach requires a function type void(T)");

        if (!this->isEmpty()) {
            this->left().forEach(f);

            f(this->front());
            this->right().forEach(f);
        }
    }

    set<T> insert(T x) const
    {
        this->assert1();

        if (this->isEmpty())
            return set(R, set(), x, set());

        T y = this->front();

        if (x == y)
            return *this;

        Color c = this->rootColor();

        if (this->rootColor() == B) {
            if (x < y)
                return balance(this->left().insert(x), y, this->right());

            return balance(this->left(), y, this->right().insert(x));
        } else {
            if (x < y)
                return set(c, this->left().insert(x), y, this->right());

            return set(c, this->left(), y, this->right().insert(x));
        }
    }

    set inserted(T x) const
    {
      auto s = insert(x);

      return set(B, s.left(), s.front(), s.right());
```

```
}

virtual bool contains(T x) const
{
    if (this->isEmpty())
        return false;

    T y = this->front();

    if (x < y)
        return left().contains(x);
    else if (y < x)
        return right().contains(x);
    else
        return true;
}

bool operator==(const set &rhs)
{
    bool ret = true;

    forEach([&ret, &rhs](T t) {
        if (!rhs.contains(t))
            ret = false;
    });

    return ret;
}

bool operator > (const set &rhs)
{
    return get_root().get()->_hash > rhs.get_root().get()->_hash;
}

bool operator < (const set &rhs)
{
```

```
            return get_root().get()->_hash < rhs.get_root().get()->_hash;
        }

        string str() const
        {
            stringstream os;

            os << "[ ";

            forEach([&os](T v) {
                os << v << " ";
            });

            os << "]";

            return os.str();
        }

        friend ostream &operator<<(std::ostream& os, const set &s)
        {
            return os << s.str();
        }
    };
}

#endif
```

8.2.7 collection.hpp

```
#ifndef __COLLECTION_HPP__
#define __COLLECTION_HPP__

#include <iostream>
#include <sstream>
#include <atomic>
#include <memory>
#include <stdexcept>
#include <cassert>
#include <functional>
#include <initializer_list>
#include <stdlib.h>
#include <time.h>

using namespace std;

namespace immut {
    enum Color { R, B };
    enum Type {L, S, M};

    template <typename T>
    class collection {
        Type type;
    public:
        Type getType() const { return type; }

        virtual bool isEmpty() const = 0;

        virtual bool contains(T x) const = 0;
    };
}

#endif
```

8.2 Test Suite Files

```
 ==> test.sh <==
#!/bin/bash

# Path to the LLVM interpreter
CXXCOMPILE="clang++ -Wall -pedantic -fsanitize=address -std=c++1y -O2
-I/usr/local/include/ -L/usr/local/lib/"
# Colors
RED='\033[0;31m'
GREEN='\033[0;32m'
CYAN='\033[0;36m'
NC='\033[0m' # No Color


# Globals
oscar_compile="./oscar -c -O"
TEST_DIR=$(pwd)

case_passed=0
case_failed=0

errorFile=errors.log
compiler_error_flag=0

# Set time limit for all operations
ulimit -t 30


show_result() {
  if [ $? -eq 0 ]; then
    echo -e "$2 passed" >> session_file
    echo -e "${GREEN}$2 passed${NC}"

    echo "" >> session_file
    echo ""

    ((case_passed++))

  else
    echo -e "$2 failed $" >> session_file
    echo -e "${RED}$2 failed${NC}"

    echo ""

    #print out expected output and result

    if [ $compiler_error_flag -eq 0 ]; then
      cat $1$2$4 >> session_file
    else
      cat $1$2$3 >> session_file
    fi
    echo "" >> session_file
    echo "Generated Output:" >> session_file
    cat ${2}_test_output  >> session_file
    echo "" >> session_file

    ((case_failed++))
  fi
}

# test pretty printing of AST
test_scanner() {
  echo "*********************************************" >> session_file
  echo "Oscar Test Script Results:" >> session_file


  for oscar_test_file in $1*.oscar ; do
    filename=$(basename "$oscar_test_file")

    echo "Testing Scanner: $filename" >> session_file
    echo "==================================" >> session_file

    # Create file to be tested (with tokens)
    $oscar_compile -s $oscar_test_file > oscar_test_output

    #Test output differences use the diff command and neglect screen output
    # diff $oscar_test_output $oscar_test_file$compiler_extension > /dev/null

    show_result $1 $filename $scanner_extension
  done
}

# test files and make sure we get the right outputs
test_compiler() {
  # set flag to prevent
  # compiler_error_flag=1

  for oscar_test_file in $1*.oscar ; do
    filename=$(basename "$oscar_test_file")
    filename="${filename%.*}"

    echo "==================================" >> session_file
    echo "Testing Compiler: $filename" >> session_file


    echo -e "Oscar Compiler Messages:" >> session_file
    $oscar_compile $oscar_test_file 2> oscar_error_output

    echo "" > oscar_test_output

    # if we had a error, then diff errors. Otherwise, run LLVM and diff outputs
    if [ -s oscar_error_output ]; then
      echo "" >> oscar_error_output
      cat oscar_error_output >> session_file
      echo "" >> session_file
```

```
          # diff errors
         diff oscar_error_output "$test_path"$filename$compiler_extension >> /dev/null
         show_result $1 $filename $compiler_extension $test_extension

       else

         # run clang on the file and save the output and errors
         echo -e "Clang Messages:" >> session_file
         $CXXCOMPILE $test_path$filename$cpp_extension 2>> session_file
         ./a.out 1> oscar_test_output 2>> session_file
         rm $test_path$filename$cpp_extension
         if [ -f $test_path$filename ]; then
           rm $test_path$filename
         fi
         echo "" >> session_file

         # diff outputs
         diff oscar_test_output "$test_path"$filename$compiler_extension >> /dev/null
         show_result $1 $filename $compiler_extension $test_extension

       fi

       done

   echo "" >> session_file
}

make_oscar(){
  echo "Oscar Compiler"
  cd ../
  make all
  echo "Oscar Compiler created"
  echo ""
}


#-----------Script starts flag checking here ------------------

echo "Oscar test started"
make_oscar

logFile=./test/logfile.log
echo "" > $logFile

echo -e "\n\n${CYAN}----------Testing Valid----------${NC}\n"
test_path=./test/oscar/compiler/
test_extension=.oscar
cpp_extension=.cpp
compiler_extension=$test_extension.out
test_compiler $test_path $compiler_extension $test_extension $cpp_extension

echo -e "\n\n${CYAN}----------Testing Errors----------${NC}\n"
# testing errors
```

```
test_path=./test/oscar/compiler/errors/
test_compiler $test_path $compiler_extension $test_extension

# cat session_file
cat session_file >> "$logFile"

#Test status output
echo ""
echo -e "Tests Passed: $case_passed $"
echo -e "Tests Failed: $case_failed $"

#Clean up temp files
rm -f oscar_test_output;
rm -f oscar_error_output;
rm -f session_file;
rm -f temp.ll;
rm a.out;

make clean
cd test
exit 0


 ==> AND.oscar <==
def main() => unit = {
  Println(true && true);
  Println(true && false);
  Println(false && true);
  Println(false && false);
}

==> AND.oscar.out <==
true
false
false
false

==> arithBinOps.oscar <==
def main() => unit = {
  Println(1 + 5);
  Println(1.0 + 2.0);
  Println(2 - 5);
  Println(3 * 5);
  Println(4 / 5);
  Println(1.0 / 3.0);
  Println(5 % 5);
}

==> arithBinOps.oscar.out <==
6
3.0000000000
-3
15
```

```
0
0.3333333333
0

==> arithFunctionChaining.oscar <==
def f1(num: int) => int = {
  return num * 2;
}

def f2(num: int) => int = {
  return num + 5;
}

def f3(num: int) => int = {
  return num - 20;
}

def main() => unit = {
    Println(f1(f2(f3(20))));
}

==> arithFunctionChaining.oscar.out <==
10

==> arithUOps.oscar <==
def main() => unit = {
  Println(17);
  Println(-42.0);
}

==> arithUOps.oscar.out <==
17
-42.0000000000

==> binOpsTypes.oscar <==
def main() => unit = {
  Println(5.0 == 5.0);
  Println(5.0 == 5.1);
  Println(5.0 != 5.1);
  Println(5.0 != 5.0);
  Println(5.0 <= 5.1);
  Println(5.1 <= 5.0);

  Println('a' == 'a');
  Println('a' == 'b');

  Println(true == true);
  Println(true == false);
}

==> binOpsTypes.oscar.out <==
true
false
true
```

```
false
true
false
true
false
true
false

==> bitwiseOps.oscar <==
def main() => unit = {
      Println(1 & 5);
      Println(1 | 4);
      Println(1 ^ 5);
      Println(5 >> 1);
      Println(5 << 1);
}

==> bitwiseOps.oscar.out <==
1
5
4
2
10

==> boolMutable.oscar <==
message testNow()


actor Tester() {
  mut bool b = false;

  receive = {
    | testNow() => {
        Println(b);
        b = true;
        Println(b);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> boolTester = spawn actor<Tester>();
  message<testNow>() |> boolTester;
}

==> boolMutable.oscar.out <==
false
true

==> bool.oscar <==
def main() => unit = {
  bool t = true;
  bool f = false;
  Println(t);
  Println(f);
}

==> bool.oscar.out <==
true
false

==> boolPrint.oscar <==
def main() => unit = {
  Println(true);
  Println(false);
  Println(5 == 5);
  Println(4 == 5);
}

==> boolPrint.oscar.out <==
true
false
true
false

==> charMutable.oscar <==
message testNow()


actor Tester() {
  mut char c = 'a';

  receive = {
    | testNow() => {
        Println(c);
        c = 'b';
        Println(c);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> charTester = spawn actor<Tester>();
  message<testNow>() |> charTester;
}

==> charMutable.oscar.out <==
'a'
'b'

==> char.oscar <==
def main() => unit = {
  Println('c');
}

==> char.oscar.out <==
c

==> charPrint.oscar <==
def main() => unit = {
  Println('a');
  Println('b');
  Println('c');
}

==> charPrint.oscar.out <==
a
b
c

==> comparisonBinOps.oscar <==
def main() => unit = {
  Println(5 == 5);
  Println(4 == 5);
  Println(4 != 5);
  Println(5 != 5);
  Println(4 < 5);
  Println(5 < 4);
  Println(5 <= 5);
  Println(5 <= 4);
  Println(5 > 4);
  Println(4 > 5);
  Println(5 >= 5);
  Println(4 >= 5);
}

==> comparisonBinOps.oscar.out <==
true
false
true
false
true
false
true
false
true
false
true
false

==> divide.oscar <==
def main() => unit = {
  Println(20 / 5);
}

==> divide.oscar.out <==
4

==> doesNotEqual.oscar <==
def main() => unit = {
  Println(1 != 2);
  Println(1 != 1);
}

==> doesNotEqual.oscar.out <==
true
false


int main () {
___monitor = new Monitor();

Println(true);

usleep(1000);
while (!___monitor->is_exitable()) {cout << "";}
delete ___monitor;
return 0;
}


==> doubleAddition.oscar <==
def main() => unit = {
  Println((1.1 + 2.2) > 3.299999 && (1.1 + 2.2) < 3.300001);
}

==> doubleAddition.oscar.out <==
true

==> doubleDivision.oscar <==
def main() => unit = {
  Println((1.1 / 2.) > 0.549999 && (1.1 / 2.) < 0.550001);
}

==> doubleDivision.oscar.out <==
true

==> doubleMultiplication.oscar <==
def main() => unit = {
  Println((1.5 * 1.5) > 2.249999 && (1.5*1.5) < 2.250001);
}

==> doubleMultiplication.oscar.out <==
true

==> doubleMutable.oscar <==
message testNow()


actor Tester() {
  mut double x = 0.0;

  receive = {
    | testNow() => {
        Println(x > -0.000001);
        x = x - 0.00001;
        Println(x > -0.000001);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> doubleTester = spawn actor<Tester>();
  message<testNow>() |> doubleTester;
}

==> doubleMutable.oscar.out <==
true
false

==> doubleSubtraction.oscar <==
def main() => unit = {
  Println((1.1 - 0.2) > 0.899999 && (1.1 - 0.2) < 0.900001);
}

==> doubleSubtraction.oscar.out <==
true

==> doubleToInt.oscar <==
def main() => unit = {
  double x = 4.0;
  Println(AsInt(x));
}

==> doubleToInt.oscar.out <==
4

==> else.oscar <==
def main() => unit = {
  if (1 > 2) {
    Println("if2 broken");
  } else {
    Println("if2 verified");
  }
}

==> else.oscar.out <==
if2 verified

==> equals.oscar <==
def main() => unit = {
  Println(1 == 1);
  Println(1 == 2);
}

==> equals.oscar.out <==
true
false

==> funcCall.oscar <==
def foo() => int = {
      return 17;
}

def main() => unit = {
      Println(foo());
}

==> funcCall.oscar.out <==
17

==> funcFormalsDouble.oscar <==
def foo(d: double) => unit = {
      Println(d);
}

def main() => unit = {
      foo(42.0);
}

==> funcFormalsDouble.oscar.out <==
42.0000000000

==> funcFormalsInt.oscar <==
def foo(a: int) => unit = {
      Println(a);
}

def main() => unit = {
      foo(17);
}

==> funcFormalsInt.oscar.out <==
17

==> funcFormalsString.oscar <==
def foo(s: string) => unit = {
      Println(s);
}

def main() => unit = {
      foo("brug");
}

==> funcFormalsString.oscar.out <==
brug

==> gcd.oscar <==
def gcd(a: int, b: int) => int = {
  if (b == 0) {
    return a;
  } else {
    return gcd(b, a % b);
  }
}

def main() => unit = {
  Println(gcd(135, 17));
  Println(gcd(40, 10));
}

==> gcd.oscar.out <==
1
10

==> greaterThanOrEqualTo.oscar <==
def main() => unit = {
  Println(2 >= 1);
  Println(1.0 >= 2.0);
}

==> greaterThanOrEqualTo.oscar.out <==
true
false

==> greaterThan.oscar <==
def main() => unit = {
  Println(2 > 1);
  Println(1.0 > 2.0);
}

==> greaterThan.oscar.out <==
true
false

==> helloWorld.oscar <==
def main() => unit = {
  Println("Hello, World!");
}

==> helloWorld.oscar.out <==
Hello, World!

==> ifElse2.oscar <==
def main() => unit = {
  if (false) {
      Println("not brug");
  } else {
    Println("brug");
  }
}

==> ifElse2.oscar.out <==
brug

==> ifElse3.oscar <==
def main() => unit = {
  if (false) {
      Println("not brug");
  } else {
      Println("brug");
    if (true) {
      Println("brug");
    }
  }
}

==> ifElse3.oscar.out <==
brug
brug

==> ifElse.oscar <==
def main() => unit = {
  if (true) {
      Println("brug");
  }
}

==> ifElse.oscar.out <==
brug

==> if.oscar <==
def main() => unit = {
  if (2 > 1) {
    Println("if1 verified");
  }
}

==> if.oscar.out <==
if1 verified

==> intMutable.oscar <==
message testNow()


actor Tester() {
  mut int x = 0;

  receive = {
    | testNow() => {
        Println(x);
        x = x + 1;
        Println(x);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> intTester = spawn actor<Tester>();
  message<testNow>() |> intTester;
}

==> intMutable.oscar.out <==
0
1



==> intToDouble.oscar <==
def main() => unit = {
  int x = 4;
  Println(AsDouble(x));
}

==> intToDouble.oscar.out <==
4.0000000000

==> lambda.oscar <==
def apply(f: [(int) => int], input: int) => int = {
  return f(input);
}

def main() => unit = {
  Println(apply((x:int) => int = x - 10, 11));

  int t = 5;

  def test() => unit = {
    def testest() => unit = {
      int b = 10;
      Println(b);
      Println(t);
    }
    testest();
    int b = 12;
    Println(b);
  }

  test();
}

==> lambda.oscar.out <==
1
10
```

```
5
12

==> lessThanOrEqualTo.oscar <==
def main() => unit = {
  Println(1 <= 2);
  Println(2.0 <= 1.0);
}

==> lessThanOrEqualTo.oscar.out <==
true
false

==> listAppend.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(Append(6, intList));
}

==> listAppend.oscar.out <==
[ 1 2 3 4 5 6 ]

==> listContains.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(Contains(1, intList));
  Println(Contains(0, intList));
}

==> listContains.oscar.out <==
true
false

==> listFilter.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(Filter((x:int) => bool = (x % 2) == 0, intList));
}

==> listFilter.oscar.out <==
[ 2 4 ]

==> listFoldLeft.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(FoldLeft((x:int, y:int) => int = {
    return x + y;
  }, 10, intList));
}

==> listFoldLeft.oscar.out <==
25

==> listForeach.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  ForEach((x:int) => unit = {
    Println(x);
  }, intList);
}

==> listForeach.oscar.out <==
1
2
3
4
5

==> listIndex.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(intList[0]);
}

==> listIndex.oscar.out <==
1

==> listMap.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(Map((x:int) => int = {
    return x + 2;
  }, intList));
}

==> listMap.oscar.out <==
[ 3 4 5 6 7 ]

==> listMutable.oscar <==
message testNow()


actor Tester() {
  mut list<int> l1 = list<int>[1, 2, 3, 4, 5];
  list<int> l2 = list<int>[0, 2, 3, 4, 5];

  receive = {
    | testNow() => {
        Println(l1 == l2);
        list<int> l3 = (l1[0] = 0);
        Println(l1 == l2);
        Die();
      }
    | die() => {
      }
  }
}
```

```
def main() => unit = {
  actor<Tester> listTester = spawn actor<Tester>();
  message<testNow>() |> listTester;
}

==> listMutable.oscar.out <==
false
true

==> list.oscar <==
def main() => unit = {
  Println(list<int>[]);
  Println(list<int>[1, 2, 3, 4, 5]);
  Println(list<string>["hey", "hi", "hello"]);
}

==> list.oscar.out <==
[ ]
[ 1 2 3 4 5 ]
[ hey hi hello ]

==> listPopBack.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(PopBack(intList));
}

==> listPopBack.oscar.out <==
[ 1 2 3 4 ]

==> listPopFront.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(PopFront(intList));
}

==> listPopFront.oscar.out <==
[ 2 3 4 5 ]

==> listPrepend.oscar <==
def main() => unit = {
  list<int> intList = list<int>[1, 2, 3, 4, 5];
  Println(Prepend(0, intList));
}

==> listPrepend.oscar.out <==
[ 0 1 2 3 4 5 ]

==> listReverse.oscar <==
def main() => unit = {
  list<int> l1 = list<int>[1, 2, 3, 4, 5];
  list<int> l2 = list<int>[5, 4, 3, 2, 1];
  Println(l2 == Reverse(l1));
}
```

```
==> listReverse.oscar.out <==
true

==> localVarDeclImmut.oscar <==
def main() => unit = {
      int i = 17;
      Println(i);

      char c = 'c';
      Println(c);

      double d = 42.0;
      Println(d);

      bool b = true;
      Println(b);

      string s = "brug";
      Println(s);
}

==> localVarDeclImmut.oscar.out <==
17
c
42.0000000000
true
brug

==> logicalBinOps.oscar <==
def main() => unit = {
  Println(true && true);
  Println(true && false);
  Println(false && false);
  Println(true || true);
  Println(true || false);
  Println(false || false);
}

==> logicalBinOps.oscar.out <==
true
false
false
true
true
false

==> mapComparison.oscar <==
def main() => unit = {
  map<int, double> tMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  map<int, double> tMapCopy = map<int, double>[0 -> 1.1, -3 -> 5.3, 4 -> 5.3];
  Println(tMap == tMapCopy);
  Println(tMap == map<int, double>[0 -> 1.1, 4 -> 5.3, 0 -> 1.1, -3 -> 5.3]);
}
```

```
==> mapComparison.oscar.out <==
true
true

==> mapContains.oscar <==
def main() => unit = {
  map<int, double> tMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  Println(Contains(0, tMap));
  Println(Contains(1, tMap));
}

==> mapContains.oscar.out <==
true
false

==> mapGet.oscar <==
def main() => unit = {
  map<int, double> tMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  Println(tMap[0]);
  Println(tMap[4]);
}

==> mapGet.oscar.out <==
1.1000000000
5.3000000000

==> mapMutable.oscar <==
message testNow()


actor Tester() {
  mut map<int, double> m1 = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  map<int, double> m2 = m1;

  receive = {
    | testNow() => {
        Println(m1 == m2);
        m1[0] = 2.7;
        Println(m1 == m2);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> mapTester = spawn actor<Tester>();
  message<testNow>() |> mapTester;
}

==> mapMutable.oscar.out <==
true
```

```
false

==> map.oscar <==
def main() => unit = {
  map<int, double> intMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  Println(intMap == map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3]);
}

==> map.oscar.out <==
true

==> mapPut.oscar <==
def main() => unit = {
  map<int, double> tMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  Println(Put(2, -1.3, tMap) == map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3, 2
-> -1.3]);
  Println(tMap == map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3]);
  Println(Put(0, 1.1, tMap) == map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3]);
}

==> mapSize.oscar <==
def main() => unit = {
  map<int, double> tMap = map<int, double>[0 -> 1.1, 4 -> 5.3, -3 -> 5.3];
  Println(Size(tMap));
  Println(Size(map<int, double>[]));
}

==> mapSize.oscar.out <==
3
0

==> mod.oscar <==
def main() => unit = {
  Println(243 % 5);
}

==> mod.oscar.out <==
3

==> multiply.oscar <==
def main() => unit = {
  Println(4 * 5);
}

==> multiply.oscar.out <==
20

==> nestedFuncs.oscar <==
def main() => unit = {
  def test(a: int) => unit = {
    Println(a);
  }

  test(5);
  int b = 4;
  Println(b);
}

==> nestedFuncs.oscar.out <==
5
4

==> NOT.oscar <==
def main() => unit = {
  Println(!false);
  Println(!true);
  Println(!true && !false);
  Println(!true || !false);
}

==> NOT.oscar.out <==
true
false
false
true

==> OR.oscar <==
def main() => unit = {
  Println(true || true);
  Println(true || false);
  Println(false || true);
  Println(false || false);
}

==> OR.oscar.out <==
true
true
true
false

==> pingPong.oscar <==
message startMsg()
message stopMsg()
message pingMsg()
message pongMsg()

actor Pong() {

  receive = {
    | pingMsg() => {
        Println("  pong");
        message<pongMsg>() |> sender;
      }
    | stopMsg() => {
        Println("pong stopped");
        Die();
      }
    | die() => {
      }
  }
}

actor Ping(pong: actor<Pong>, maxTurns : int) {
  mut int count = 0;

  def incrementAndPrint() => unit = {
    count = count + 1;
    Println("ping");
  }

  receive = {
    | startMsg() => {
        incrementAndPrint();
        message<pingMsg>() |> pong;
      }
    | pongMsg() => {
        incrementAndPrint();

        if (count > maxTurns) {
            Println("ping stopped");
            message<stopMsg>() |> pong;
            Die();
        } else {
            message<pingMsg>() |> pong;
        }
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Pong> pong = spawn actor<Pong>();
  actor<Ping> ping = spawn actor<Ping>(pong, 99);

  message<startMsg>() |> ping;
}

==> pingPong.oscar.out <==
ping
  pong
ping
  pong
ping
  pong
ping
ping stopped
pong stopped

==> pool.oscar <==
message testNow()
```

```
actor Tester() {

  receive = {
    | testNow() => {
        Println(10);
      }
    | die() => {
      }
  }
}

def main() => unit = {
  pool<Tester> testerPool = spawn pool<Tester>({}, 3);
  list<message<testNow>>[message<testNow>(), message<testNow>(),
message<testNow>()] |>> testerPool;
}

==> pool.oscar.out <==
10
10
10

==> recursion.oscar <==
message testNow()


actor Tester() {

  def factorial(n: int) => int = {
    if (n == 0) {
      return 1;
    } else {
      return (n * factorial(n - 1));
    }
  }

  receive = {
    | testNow() => {
        Println(factorial(5));
        Die();
      }
    | die() => {
      }
  }
}


def main() => unit = {
  actor<Tester> recursionTester = spawn actor<Tester>();
  message<testNow>() |> recursionTester;
}
```

```
==> recursion.oscar.out <==
120

==> scoping.oscar <==

def main() => unit = {
  int a = 5;
  {
    Println(a);
    int b = 6;
    Println(b);
  }
  Println(a);
  int b = 4;
  Println(b);
}

==> scoping.oscar.out <==
5
6
5
4

==> setAdd.oscar <==
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  Println(Put(6, intSet1) == set<int>[1, 2, 3, 4, 5, 6]);
  Println(intSet1 == set<int>[1, 2, 3, 4, 5]);
  Println(Put(1, intSet1) == intSet1);
}

==> setAdd.oscar.out <==
true
true
true

==> setComparison.oscar <==
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  set<int> intSet1Copy = set<int>[2, 1, 3, 5, 4];
  Println(intSet1 == intSet1Copy);
  Println(intSet1 == set<int>[1, 2, 3, 4, 5, 5]);
}

==> setComparison.oscar.out <==
true
true

==> setContains.oscar <==
def main() => unit = {
  set<int> intSet = set<int>[1, 2, 3, 4, 5];
  Println(Contains(3, intSet));
  Println(Contains(0, intSet));
}
```

```
==> setContains.oscar.out <==
true
false



==> setDiff.oscar <==
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  set<int> intSet2 = set<int>[0, 3, 4, 6];
  Println(Diff(intSet1, intSet2) == set<int>[1, 2, 5]);
  Println(Diff(intSet2, intSet1) == set<int>[0, 6]);
  Println(Diff(intSet1, intSet1) == set<int>[]);
}

==> setDiff.oscar.out <==
true
true
true

==> setIntersect.oscar <==
def main() => unit = {
  set<int> intSet1 = set<int>[1, 2, 3, 4, 5];
  set<int> intSet2 = set<int>[1, 2, 4, 7, 9];
  Println(Intersection(intSet1, intSet2) == set<int>[1, 2, 4]);
  Println(Intersection(intSet2, intSet1) == set<int>[1, 2, 4]);
}

==> setIntersect.oscar.out <==
true
true

==> setMutable.oscar <==
message testNow()


actor Tester() {
  mut set<int> s1 = set<int>[1, 2, 3, 4, 5];

  receive = {
    | testNow() => {
                set<int> s2 = setAdd(s1, 6);
                Println(s1 == s2);
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> setTester = spawn actor<Tester>();
  message<testNow>() |> setTester;
```

```
}

==> setMutable.oscar.out <==
true

==> set.oscar <==
def main() => unit = {
  Println(set<int>[]);
  Println(set<int>[1]);
  Println(set<int>[2, 2, 2]);
  Println(set<int>[1, 2, 3] == set<int>[3, 2, 1]);
}

==> set.oscar.out <==
[ ]
[ 1 ]
[ 2 ]
true

==> setUnion.oscar <==
def main() => unit = {
  set<int> intSet1 = set<int>[0, 1, 2, 3];
  set<int> intSet2 = set<int>[0, 4, 5];
  Println(Union(intSet1, intSet2) == set<int>[0, 1, 2, 3, 4, 5]);
}

==> setUnion.oscar.out <==
true

==> signedOps.oscar <==
def main() => unit = {
  Println(5 + -5);
  Println(5 * -5);
  Println(5 / -5);
  Println(5 - -5);
}

==> signedOps.oscar.out <==
0
-25
-1
10

==> stringConcatenation.oscar <==
def main() => unit = {
  string s = "string";
  Println("pull " + s);
}

==> stringConcatenation.oscar.out <==
pull string

==> stringIndex.oscar <==
def main() => unit = {
  string s1 = "string1";
  Println(s1[0]);
}

==> stringIndex.oscar.out <==
s

==> stringMutable.oscar <==
message testNow()


actor Tester() {
  mut string s = "unchanged";

  receive = {
    | testNow() => {
        Println(s == "changed");
        s = "changed";
        Println(s == "changed");
        Die();
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> stringTester = spawn actor<Tester>();
  message<testNow>() |> stringTester;
}

==> stringMutable.oscar.out <==
false
true

==> string.oscar <==
def main() => unit = {
  string s2 = "string2";
  Println("string1");
  Println(s2);

  string escp = "\"\'\\\\n\ttest";
  Println(escp);
}

==> string.oscar.out <==
string1
string2
"'\\n test

==> stringSize.oscar <==
def main() => unit = {
  string s1 = "string1";
  Println(Size(s1));
```

```
}

==> stringSize.oscar.out <==
7

==> subtract.oscar <==
def main() => unit = {
  Println(5 - 5);
}

==> subtract.oscar.out <==
0

==> toString.oscar <==
def main() => unit = {
  Println(AsString(123) == "123");
}

==> toString.oscar.out <==
true

==> value.oscar <==
def main() => unit = {
  int x = 4;
  int y = 10;
  Println(x);
  Println(x * -5);
  Println(x / -5);
  Println(x - -5);

  Println(x - y);
  Println(x + y);
  Println(x * y);
  Println(x / y);
}

==> value.oscar.out <==
4
-20
0
9
-6
14
40
0


==> actorAlreadyDeclared.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  message<testNow>() |> tester1;
  message<die>() |> tester1;
}

==> actorAlreadyDeclared.oscar.out <==
Error: Actor Tester declared already

==> actorEquality.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  actor<Tester> tester2 = spawn actor<Tester>();
  bool b = (tester1 == tester2);
  message<testNow>() |> tester1;
  message<die>() |> tester1;
  message<die>() |> tester2;
}

==> actorEquality.oscar.out <==
Error: Actors cannot be compared for equality
```

```
==> actorNotFound.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Lost> tester = spawn actor<Lost>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> actorNotFound.oscar.out <==
Error: Actor of type Lost not found

==> actorParamMismatch.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>(4);
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> actorParamMismatch.oscar.out <==
Error: Actor constructed with conflicting parameter types Tester requires () but
constructed with int

==> assignMismatch.oscar <==
def main() => unit = {
  int x = "1";
}
```

```
==> assignMismatch.oscar.out <==
Error: Value initialization type mismatch: x is int but initialized as string

==> dupReceivePMatching.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error1");
        }
    | testNow() => {
        Println("Error2");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  message<testNow>() |> tester1;
  message<die>() |> tester1;
}

==> dupReceivePMatching.oscar.out <==
Error: Duplicate pattern matching in receive in actor Tester

==> funcAlreadyDeclared.oscar <==
def add5(x: int) => int = {
  return (x + 5);
}

def add5(x: int) => int = {
  return (x + 5);
}

def main() => unit = {
  add5(1);
}

==> funcAlreadyDeclared.oscar.out <==
Error: Value add5 declared already

==> funcDoesNotReturn.oscar <==
def add5(x: int) => int = {
  int result = x + 5;
}

def main() => unit = {
  int x = add5(1);
}
```

```
==> funcDoesNotReturn.oscar.out <==
Error: This function must return int

==> funcNotFound.oscar <==
def add5(x: int) => int = {
      return x + 5;
}

def main() => unit = {
      int y = add5(1, 5);
}

==> funcNotFound.oscar.out <==
Error: Function add5 has signature (int) => unit but was called with args (int,
int)

==> funcReturnMismatch.oscar <==
def add5(x: int) => int = {
      return 6.0;
}

def main() => unit = {
      int y = add5(1);
}

==> funcReturnMismatch.oscar.out <==
Error: Return type mismatch: expected int but returns double

==> functionNotFound.oscar <==
def main() => unit = {
  test();
}
==> functionNotFound.oscar.out <==
Error: Function test not found

==> idNotFound.oscar <==
def main() => unit = {
  int x = y;
}
==> idNotFound.oscar.out <==
Error: Undeclared identifier y

==> immutReassignment.oscar <==
def main() => unit = {
  int x = 1;
  x = 4;
}
==> immutReassignment.oscar.out <==
Error: Reassignment to a value x

==> initValueMismatch.oscar <==
def main() => unit = {
  int x = 1.0;
}
```

```
==> initValueMismatch.oscar.out <==
Error: Value initialization type mismatch: x is int but initialized as double

==> invalidAccess.oscar <==
def main() => unit = {
  list<int> l1 = list<int>[1, 2, 3, 4, 5];
  int x = l1["0"];
}
==> invalidAccess.oscar.out <==
Error: Invalid access types list<int> cannot be accessed by string

==> invalidBroadcastOp.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  message<testNow>() |>> tester1;
  message<die>() |> tester1;
}

==> invalidBroadcastOp.oscar.out <==
Error: Invalid broadcast operation between message<testNow> to actor<Tester>

==> invalidSendOp.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  pool<Tester> pool1 = spawn pool<Tester>({}, 3);
  message<testNow>() |> pool1;
  message<die>() |>> pool1;
```

```
}

==> invalidSendOp.oscar.out <==
Error: Invalid send operation between message<testNow> to pool<Tester>

==> logicalOpMismatch.oscar <==
def main() => unit = {
  Println(1 && true);
}
==> logicalOpMismatch.oscar.out <==
Error: Only boolean expressions are allowed for &&

==> messageNotFound.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testLater>() |> tester;
  message<die>() |> tester;
}

==> messageNotFound.oscar.out <==
Error: Message of type testLater not found

==> messageNotInReceive.oscar <==
message testNow()
message testLater()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  message<testLater>() |> tester1;
  message<die>() |> tester1;
}

==> messageNotInReceive.oscar.out <==
Error: No matching pattern to receive message testLater in actor Tester

==> messageNotInReceivePool.oscar <==
message testNow()
message testLater()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  pool<Tester> pool1 = spawn pool<Tester>({}, 3);
  message<testLater>() |>> pool1;
  message<die>() |> tester;
}

==> messageNotInReceivePool.oscar.out <==
Error: No matching pattern to receive message testLater in pool of actor type
Tester

==> miscEquality.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println(1 == 1.0);
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}
```

```
==> miscEquality.oscar.out <==
Error: Cannot compare int with double

==> modMismatch.oscar <==
def main() => unit = {
  Println(3.0 % 2.0);
}
==> modMismatch.oscar.out <==
Error: operand type mismatch: double % double

==> msgAlreadyDeclared.oscar <==
message testNow()
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testLater>() |> tester;
  message<die>() |> tester;
}

==> msgAlreadyDeclared.oscar.out <==
Error: Message testNow declared already

==> msgParamMismatch.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>("error", 1) |> tester;
  message<die>() |> tester;
}

==> msgParamMismatch.oscar.out <==
Error: Message constructed with conflicting parameter types testNow requires () but
constructed with string, int

==> mutAlreadyDeclared.oscar <==
message testNow()


actor Tester() {
  mut int x = 1;
  mut int x = 2;

  receive = {
    | testNow() => {
        Println(x);
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> mutAlreadyDeclared.oscar.out <==
Error: Mutable var x declared already

==> mutImmutDecl.oscar <==
message testNow()


actor Tester() {
  mut int x = 1;
  int x = 2;

  receive = {
    | testNow() => {}
      }
    | die() => {
      }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> mutImmutDecl.oscar.out <==
Error: Value x declared already

==> mutInitTypeMismatch.oscar <==
message testNow()


actor Tester() {
  mut int x = 1.0;

  receive = {
    | testNow() => {
        Println(x);
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> mutInitTypeMismatch.oscar.out <==
Error: Variable initialization type mismatch: x is int but initialized as double

==> mutNotInActor.oscar <==
def main() => unit = {
  mut int x = 1;
}

==> mutNotInActor.oscar.out <==
Error: Mutables types are only allowed in actors

==> mutOutsideActor.oscar <==
def main() => unit = {
      mut int x = 17;
      Println(x);
}

==> mutOutsideActor.oscar.out <==
Error: Mutables types are only allowed in actors

==> mutReassignMismatch.oscar <==
message testNow()


actor Tester() {
  mut int x = 1;

  receive = {
    | testNow() => {
        x = 2.0;
      }
    | die() => {
      }
  }
}

def main() => unit = {
  actor<Tester> tester = spawn actor<Tester>();
  message<testNow>() |> tester;
  message<die>() |> tester;
}

==> mutReassignMismatch.oscar.out <==
Error: Assignment to incompatible types: double cannot be assigned to int

==> noMainFunc.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error1");
      }
    | die() => {
        }
  }
}

==> noMainFunc.oscar.out <==
Error: No main function found in this program

==> operandMismatch.oscar <==
def main() => unit = {
  Println(1 + 1.0);
}
==> operandMismatch.oscar.out <==
Error: operand type mismatch: int + double

==> poolActorNumber.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}
```

```
def main() => unit = {
  pool<Tester> pool1 = spawn pool<Tester>({}, "3");
  message<testNow>() |>> pool1;
  message<die>() |>> pool1;
}

==> poolActorNumber.oscar.out <==
Error: Number of actors must be an integer

==> poolActorUndecl.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  pool<Worker> pool1 = spawn pool<Worker>({}, 3);
  message<testNow>() |>> pool1;
  message<die>() |>> pool1;
}

==> poolActorUndecl.oscar.out <==
Error: Actor of type Worker not found

==> poolEquality.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error");
      }
    | die() => {
      }
  }
}

def main() => unit = {
  pool<Tester> pool1 = spawn pool<Tester>({}, 3);
  pool<Tester> pool2 = spawn pool<Tester>({}, 3);
  bool b = (pool1 == pool2);
  message<testNow>() |>> pool1;
  message<die>() |>> pool1;
  message<die>() |>> pool2;
}

==> poolEquality.oscar.out <==
Error: Pools cannot be compared for equality

==> receiveMsgUndefined.oscar <==
message testNow()


actor Tester() {

  receive = {
    | testNow() => {
        Println("Error1");
          }
    | testLater() => {
        Println("Error1");
        }
  }
}

def main() => unit = {
  actor<Tester> tester1 = spawn actor<Tester>();
  message<testNow>() |> tester1;
  message<die>() |> tester1;
}

==> receiveMsgUndefined.oscar.out <==
Error: Actor Tester attempts to receive an undefined message

==> script.sh <==
Error: for f in *; do
      sed -i old '1s/^/Error: /' "$f"
done

==> undeclaredId.oscar <==
def main() => unit = {
  x = 1;
}
==> undeclaredId.oscar.out <==
Error: Undeclared identifier x

==> unitDecl.oscar <==
def main() => unit = {
  unit a = unit;
  Println(a);
}

==> unitDecl.oscar.out <==
Error: Cannot declare value of type unit to a

==> uopMismatch.oscar <==
def main() => unit = {
  string s1 = "test";
  Println(-s1);
}
==> uopMismatch.oscar.out <==
Error: operand type mismatch: - on string

==> valAlreadyDeclared.oscar <==
def main() => unit = {
  int x = 1;
  int x = 2;
}
==> valAlreadyDeclared.oscar.out <==
Error: Value x declared already


==> actor.oscar <==
actor Master(numWorkers: int, numMsgs: int, numElems: int) {
  mut pi = 0.0;
  mut numResults = 0;
  mut pool<Worker> workerPool = spawn pool<Worker>(numWorkers);

  Listener listener = spawn Listener("pi listener");

  receive {
    | start() => {
        list<work> msgList = [int i <- 0 to numMsgs by 1].map(i => {
          work(i * numElems, numElems);
        })

        msgList |>> workerPool;
      }
    | result() => {
        pi = pi + value;
        numResults = numResults + 1;
        if (numResults == numMsgs) {
          piApproximation(pi) |>> listener;
        }
      }
    | end => {
        die();
      }
  }
}
==> actor.oscar.scan <==

==> chr_str.oscar <==
def main() => unit = {
  char a = ' ';
  char b = '#';
  char c = ')';
  char d = '~';
  char e = 'a';
  char f = 'A';
  char g = '1';
  char h = '\t';
  char n = '\n';
  char ap = '\'';
  char qo = '\"';
  char s = '\\';

  string test =
"1234567890qwertyuiopasdfghjklzxcvbnm[{]};:'",<.>/?!@#$%^&*()_+-=`~\t\n\r\\\'\"";
}

==> gen_pi.oscar <==
message start()
message end()
message work(start: int, numElems: int)
message result(value: double)
message piApproximation(pi: double)


actor Worker() {
  def genRange(start: int, end: int) => list<int> = {
    return Reverse(MergeFront(genRange(start + 1, end), list<int>[start]));
  }

  def calcPi(start: int, numElems: int) => double = {
    list<int> range = genRange(start, start + numElems);

    return Reduce((x: double, y: double) => double = {return x + y;},
      Map((i: int) => double = {
        return 4.0 * AsDouble((1 - (i % 2) * 2) / (2 * i + 1));
      }, range));
  }

  receive = {
    | work(start: int, numElems: int) => {
        double pi = calcPi(start, numElems);

        message<result>(pi) |> sender;
      }
  }
}

actor Listener() {
  receive = {
    | piApproximation(value: double) => {
        Println("value of pi is approximately :" + AsString(value));
        message<end>() |> sender;
    }
  }
}


actor Master(numWorkers: int, numMsgs: int, numElems: int) {
  mut double pi = 0.0;
  mut int numResults = 0;
  pool<Worker> workerPool = spawn pool<Worker>({}, numWorkers);
  actor<Listener> listener = spawn actor<Listener>();

  receive = {
    | start() => {
        /*list<work> msgList = [int i <- 0 to numMsgs by 1].map(i => {
          work(i * numElems, numElems);
        })*/
        message<work> msg = message<work>(0, 10);
        msg |>> workerPool;
      }
    | result(value: double) => {
        pi = pi + value;
        numResults = numResults + 1;
        if (numResults == numMsgs) {
          message<piApproximation>(pi) |> listener;
        }
      }
    | end() => {
        die();
      }
  }
}


def main() => unit = {
  actor<Master> master = spawn actor<Master>(5, 10000, 10000);
  message<start>() |> master;
}

==> gen_pi.oscar.scan <==
message start()
message end()
message work(start: int, numElems: int)
message result(value: double)
message piApproximation(pi: double)

actor Worker() {


def genRange(start: int, end: int) => list<int> = {
return listReverse(listPrepend(genRange((start + 1), end), list<int>[start]));
}

def calcPi(start: int, numElems: int) => double = {
list<int> range = genRange(start, (start + numElems));
return listReduce((x: int, y: int) => int = {
return (x + y);
}, listMap((i: int) => int = {
((4. * (1 - ((i % 2) * 2))) / ((2 * i) + 1));
}, range));
}
```

```
receive = {
| work(start: int, numelems: int) => {
double pi = calcPi(start, numElems);
message<result>(pi) |> sender;
}
}
}

actor Listener() {




receive = {
| piApproximation(value: double) => {
println(("value of pi is approximately :" + value));
message<end>() |> sender;
}
}
}

actor Master(numWorkers: int, numMsgs: int, numElems: int) {
mut double pi = 0.;
mut int numResults = 0;
mut pool<Worker> workerPool = spawn pool<Worker>({}, numWorkers);
actor<Listener> listener = spawn actor<Listener>("pi listener");



receive = {
| start() => {
message<work> msg = message<work>(0, 10);
msgList |>> workerPool;
}
| result(value: double) => {
(pi = (pi + value));
(numResults = (numResults + 1));
if ((numResults == numMsgs)) {
message<piApproximation>(pi) |>> listener;
}
}
| end() => {
die();
}
}
}

def main() => unit = {
actor<Master> master = spawn actor<Master>(5, 10000, 10000);
message<start>() |> master;
}

==> lambda.oscar <==
def apply(f: [(int) => int], input: int) => int = {
  return f(input);
}

def main() => unit = {
  int a = 12;
  int b = apply((x:int) => int = x - 10, a);

  Println(b);
}
```

```
def AsString(b : bool) => string = {
if (b) {
return "true";
} else {
return "false";
}
}

func apply = (f : [(int) => int], input : int) => int = f(input);

def main() => unit = {
int a = 12;
int b = apply((x : int) => int = (x - 10), a);
Println(b);
}

==> types.oscar <==
/*
 * COMMENT BLOCK
 */

actor Types() {
  int a = 4;
  int pos_i = 1;
  int neg_i = -1;
  double d = 2.5;
  bool b_t = true;
  bool b_f = false;
  char c = 'a';
  string s = "test";
  list<int> intList = list<int>[1, 2, 3];
  mut list<list<int> > intListList = list<list<int> >[intList];

  set<int> intSet = set<int>[1, 2, 3];
  map<int, char> intCharMap = map<int, char>[1 -> 'a', 2 -> 'b'];

  int accList = intList[2];
  char accMap = intCharMap[3];

  mut int mI;
  mI = 2;
  mut double mD;
  mD = 5.0;
  mut bool mB;
  mB = true;
  mB = false;
  mut char mC;
  mC = 'b';
  mut string mS;
  mS = "mut test";
  mut list<int> mIntList;
  mut set<int> mIntSet;
  mut map<int, char> mICM;

  intListList[0][0] = mI;
}

def main() => unit = {
  Println("Hello World!");
}
```

9 Git Log

Our git history is a bit messed up as we decided to squash our commits rather than leave them as they are.

```
commit 960327c95ecff5739ae7f2450bd883e42b1540ba
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:51:36 2016 -0500

    deleted cpp stuff

commit 242e6359c670a82fec0eb1de496fb3bfb8f75f36
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:50:45 2016 -0500

    removed useless crap

commit 1779a19f8fddc88e064d76603f305381f1477791
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:45:09 2016 -0500

    fixed the thing

commit a5d57e647ff512f0dada300cef7a9639a87fec63
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:30:53 2016 -0500

    actors work properly!

commit ba2b9f0125ac1fdc73dd95397048df66149961aa
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:30:06 2016 -0500

    added sleep, slightly cleaner

commit db15489d70ddc5b9262a12985c18f914678355c8
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 23:06:11 2016 -0500

    up to date

commit c1ca43c9b103678d6d33ab3bc5fbbed7e0c24d45
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 22:44:18 2016 -0500

    welp

commit 4cb8c4b15802b71dca7347332e1b68626639d5fd
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 22:40:14 2016 -0500
```

```
    pingPong different

commit 3e8101b6828a84f6084e44c9c12e13b32cd94c47
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 21:37:42 2016 -0500

    runs without memory leaks... but doesn't halt

commit b3fa2e0d123cc725e34c664369ba5541d034e4b9
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 21:31:02 2016 -0500

    removing compiled stuff

commit 647f6b4fabcd1a60c161eebc014c7d101e2cf937
Merge: b9f1595 fc81f00
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 17:20:01 2016 -0500

    Merge pull request #132 from Howon/actor

    Actor

commit fc81f001d13eb40b560adebe4cfe8f32317c4ef0
Merge: 481959e b9f1595
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 17:17:19 2016 -0500

    demo code

commit 481959ec694f7ac328917fbcc6ecae9b84c7f0c9
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 10:50:04 2016 -0500

    demo stuff

commit b9f1595a00d2b78fede2e5e1b62a8f1f746aa409
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 10:49:27 2016 -0500

    removed flags, added actor stuff

commit 6937f93a19e1c4f612ada9e64dfaddb982093c48
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 10:46:39 2016 -0500

    Actor (#131)

    * almost there for actors

    * killing Pong properly

    * plugging memory leaks
    * plugging memory leaks #2

    * added delete

    * no more memory leaks in actors :)

    * fixed making and top-level to be less verbose/able to make without sudo

    * fixed optimizer to match howon's changes

    * removed cpps in test

    * adding pthread

    * enforcing a die on all actors

    * enforcing a die on all actors

    * added basic thing

    * added basic

    * more codegen

    * attempting monitor. someone try fixing this

    * someone try fixing this deadlock

    * fixed a bunch of tests

    * fixed all tests with actors

    * oops removed some cpp stuff

    * seems to terminate now

commit b0e100ab39f101c7ab76fec7c35164310af30bba
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 08:53:39 2016 -0500

    seems to terminate now

commit e2a9df49a0399640a9ffd5521f9011175c09d83a
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 05:06:57 2016 -0500

    oops removed some cpp stuff

commit ffe97c528ae42a24c8862e787c359334376efad2
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 05:05:29 2016 -0500

    fixed all tests with actors
```

```
commit cd0a5b861c6ee0dea394e4ec2382b47f54a7e218
Merge: dcc209e 5e17cfc
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 04:53:02 2016 -0500

    Merge branch 'actor' of github.com:Howon/Oscar into actor

commit dcc209e8ba699c0bb8c5997accf5c35718b9a3e7
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 04:52:00 2016 -0500

    fixed a bunch of tests

commit 5e17cfc6afe1f4ffb19eac13f73fa8cbd6a83cf8
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 04:51:57 2016 -0500

    someone try fixing this deadlock

commit 8b5e2c89e754e1a84e0ad9a93c1c8d67e0ad572a
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 20 04:44:13 2016 -0500

    attempting monitor. someone try fixing this

commit 27247b94d6aa6cc944c2085858914a516ccf41d7
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 03:59:18 2016 -0500

    more codegen

commit 0ed61509175d470b2436fbab1fb20ae91b3adab7
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 02:49:47 2016 -0500

    added basic

commit 3b9be279b3d0eaae03c8b77702944dac65b95878
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 02:48:12 2016 -0500

    added basic thing

commit 6bed6e080f69d7daac612b7c8136545ff22b02f0
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 02:43:09 2016 -0500

    enforcing a die on all actors

commit 59b00635b57f1a123f813c6a605dbdf64af79496
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 02:42:36 2016 -0500
```

```
    enforcing a die on all actors

commit 4796338ff29f3662374869a5fbe16ec5f3a48b0c
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 02:17:34 2016 -0500

    adding pthread

commit 6cfb33d601e6546ebf305201fcd547a9a80053a7
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 00:58:23 2016 -0500

    removed cpps in test

commit 4fcb15cd5adf56b5faee534bbbc03e78d8bed7f2
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 00:57:01 2016 -0500

    fixed optimizer to match howon's changes

commit 55ab8c3b05a358223129b02fd5373fe6ce3773fa
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 00:56:45 2016 -0500

    fixed making and top-level to be less verbose/able to make without sudo

commit adddeed72c37b7c506e730cc32b316fd547c58a5
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 20 00:38:41 2016 -0500

    no more memory leaks in actors :)

commit b948e40d19700d678833877828fb032ef297d2dc
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 23:26:03 2016 -0500

    added delete

commit e1ed46c971bfeccbba10038866ba7f2fb518fd98
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 23:15:32 2016 -0500

    plugging memory leaks #2

commit 5b51aa7920a1ae6802566b39683d7d66e5474d1f
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 23:13:07 2016 -0500

    plugging memory leaks

commit 8969a5217eb996753638931bcab9ee307df9bf9d
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 22:59:45 2016 -0500
```

```
    killing Pong properly

commit 26381980a8956c3918e51e26b763893c6bbb901d
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 22:21:29 2016 -0500

    almost there for actors

commit 2f28aee671c22690d7079562cc2979bb88c3ab6b
Merge: 76c9cb9 ecd1764
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 18:21:39 2016 -0500

    Merge pull request #128 from Howon/vs/pingpong_oscar

    adding Oscar-ized Ping Pong test

commit ecd1764ac21830c3dcdbf0c575e705560999f6c6
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Mon Dec 19 18:19:24 2016 -0500

    pingPong now passes analyzer

commit ee17abea90b5bdf2a2176870c7657afb23289bac
Merge: 752bba1 76c9cb9
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Mon Dec 19 18:16:17 2016 -0500

    Merge branch 'master' into vs/pingpong_oscar

commit 76c9cb9390f3807a830d47098344a9776b7e83cd
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Mon Dec 19 18:13:43 2016 -0500

    Fix receive (#130)

    * analyzer receive analysis maybe working

    * newline after errors

    * think I fixed the issue with receive

commit 2957f43f45b354c9ec5d71349aa393f80dcadd0a
Merge: 03ae6ac 8d68438
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 17:11:06 2016 -0500

    Merge pull request #129 from Howon/actr-compile

    Actr compile

commit 752bba1abccdc19225cbd0ac93e6dc58fa6a8d3f
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Mon Dec 19 16:47:24 2016 -0500
```

```
    fixed pingPong syntax

commit 8d68438d4e9bafc394dcfc5bc5a989c36b9d2b91
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 16:37:52 2016 -0500

    explicit die() call allowed now

commit 28217da07ec585f1b87c915843e95cb3a4a0a66f
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 16:22:56 2016 -0500

    try this

commit c83244952f18ad266484e3ae669dd422f6c20795
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 16:20:04 2016 -0500

    adding Oscar-ized Ping Pong test

commit febe4bcb5b01428e6241636bf7dda9d346cf1607
Merge: 9c81886 03ae6ac
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 11:48:20 2016 -0500

    Merge branch 'master' of https://github.com/Howon/Oscar

commit 9c818862475502f541b49c62197371b3927eecef
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 11:48:17 2016 -0500

    changed file names

commit 03ae6ac1d6cd78a269a11add1d776017e4909522
Merge: 3a48269 78adc87
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 19 11:47:54 2016 -0500

    Merge pull request #114 from Howon/vs/actors_cpp

    Actor prototype

commit 78adc8763be49a8345dfc0e488e755d8f219ffd0
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 11:17:21 2016 -0500

    separated PingPong into its own class

commit 5601ee0bd5d309f0899e88012a916e7d4591f980
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 10:24:16 2016 -0500

    more code clean up
```

```
commit 4150423fe7db6f9f716370682c470f6ab04754e3
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 10:16:01 2016 -0500

    plugged memory leaks

commit f7dadc02cbdbf0bf305dcc51431b05d83272c741
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 10:04:34 2016 -0500

    stopping threads when finished executing

commit 131025a1532eb29cd7135a3ded7ce4dfc26b6fcb
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 01:53:14 2016 -0500

    code cleaned up

commit 0ab745741bd33f8b4a5fa0051406af2a30c11935
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 01:49:24 2016 -0500

    ping-pong, ping-pong, pingy-pongy pong pong pong

commit 3a48269e5d6e1544393480771b1c4ac130c0c4f3
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Mon Dec 19 00:24:06 2016 -0500

    Jh/opt strings (#127)

    * strings optimized, fixed transpile return

    * oscar slightly less error prone

commit 87fcfa96ccd341ecb65db83cfa9b1a294b97011a
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Mon Dec 19 00:09:53 2016 -0500

    WIP: ping pong, first 'ping' printed, then hangs

commit 34b05c2e59bb0ee8fe30937ab784b8b7abb4103d
Merge: d3decb9 605fccf
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 18 21:24:28 2016 -0500

    Merge pull request #124 from Howon/fix_toplevel_transpile

    Fix toplevel transpile

commit d3decb9deb701555af4270bc443814c498c8f32e
Merge: bea7e04 a1ebded
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 18 21:24:14 2016 -0500
```

```
    Merge pull request #125 from Howon/jh/secondpass

    Jh/secondpass

commit bea7e04c9b0e49b55e0aa1e61e0be5ec432dc28b
Merge: 9f731cf d37cc48
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 18 21:24:10 2016 -0500

    Merge pull request #126 from Howon/assign_analysis

    fixed issue with assignment and containers

commit d37cc4811055fc08b604758df0fc6d9d67a734cd
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 21:09:41 2016 -0500

    fixed issue with assignment and containers

commit 605fccfa545fbd5cac8086527cd388826aae68c6
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 20:55:47 2016 -0500

    finalizing

commit a98b1fab318556ebdda6b6a3b78e9db96bff69cc
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Dec 18 20:48:16 2016 -0500

    added infinite loop in the consume() method to keep listeing for messages

commit a1ebded5d57a6bc1aa4e812202ec1c18e42d9403
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 20:19:34 2016 -0500

    fixed access

commit 2ddc7bbc84e8c4ffed3e571c6d105270ca6f5964
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 20:06:16 2016 -0500

    second pass! removes unused vars!

commit 0df4b8f568c6b8b5273b636c47f6d40de2830604
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Dec 18 19:54:24 2016 -0500

    Removed pure virtual get() method from base class

commit 1ff16764a28cd48f9d9d33eacfed30f0d9a121cf
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Dec 18 19:27:30 2016 -0500
```

```
    updated prototype to include message inheritance and passing

commit bf21f168dc09088c0f53be13f65de4736a7e4c11
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 19:17:35 2016 -0500

    fixing minor things

commit 078710d8b785176dc8ee674419a439cae0e0b544
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 19:11:20 2016 -0500

    finished

commit 52ebb628a8f8f3b7cd707d5b8f249b063267b189
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 18:46:35 2016 -0500

    fixing

commit cb342f9259261498f41f0f6067ec04f643fb9d14
Merge: bc3d3bb 9f731cf
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 18:43:42 2016 -0500

    fixing toplevel again

commit 9f731cfb911114952d0b62d5aa8c2640b53ffe3e
Merge: f55d31a 9b6f873
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 18 18:11:37 2016 -0500

    Merge pull request #123 from Howon/fixfailingtests

    Fixfailingtests

commit bc3d3bb74009d4b9be04d43ad7f8beb9d4591e7f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 18:07:19 2016 -0500

    moving test to where it should be

commit 9b6f87352a57eddba48cce8f78cda1389736832b
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 18:05:24 2016 -0500

    only 13 test fail!

commit b957d985c535044f31cbad1e1961c4d941d636b9
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Dec 18 17:38:52 2016 -0500

    fixing some tests
```

```
commit 80c257d87bad7d736ea5dea55ff4639787da5e10
Merge: d904182 f55d31a
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 17:13:10 2016 -0500

    Merge branch 'master' into fixfailingtests

commit d9041829d0f7bacde7af1f684dc4287e5451250c
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 18 17:12:56 2016 -0500

    fixed failing tests

commit f55d31a70ee6b4aed93604964348cf841181902c
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Sun Dec 18 17:04:36 2016 -0500

    Jh/optimize (#122)

    * single pass optimizer working

    * single pass complete

    * bit more work to do on second pass but I want to sleep

    * scoping is better, replacing functions though

    * single pass optimizer looking good

    * fixed bool comparisons

    * fixed list lits

    * test uses optimizer

commit f983ffe16480747e92fdfb8299267496424b369f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 23:04:37 2016 -0500

    toplevel

commit f74f782f1eea7c94a67105e31a752b658401a69b
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 22:58:56 2016 -0500

    fixing

commit 93e3372f541e96f9bc4ae05f534d8f5271fdbee9
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 22:49:14 2016 -0500

    fixing

commit 1cf0667cb27e4ed93c451b46dbfb5f06d0c7abfc
```

```
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 22:43:41 2016 -0500

    fixing

commit 03c8d0fceacaf1aad57591dfc3a09f6068d656b1
Merge: a957c53 3f04b4a
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 21:53:02 2016 -0500

    merge

commit 3f04b4a0a73595d62a379ecfa97c1450cc566eca
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 21:41:39 2016 -0500

    Transpile (#115)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing

    * default adds a newline to print

    * adding printing of binops

    * adding mod

    * renamed n stuff

    * updating

    * making testing testy

    * approaching

    * removed unneeded junks

    * .out files added
```

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

* Recursive functions work now

Changed Message_t, Actor_t, Pool_t

* attempt at resolving builtin container functions

* fixing sexpr

* progress on variables

* fixed locals?

* can print all types, just need to bring in analysis

* adding println definitions for more types

* spacing

* variables work for ints chars doubles and bools

* cleanup

* t_expr everywhere

* variables seem to work

```
    * renaming all tests to use Println

    * trying to make bools print right

    * fixing uop tests because we don't have bitwise negation

    * gen pi checks out

    * gcd works now

    * oops

    * fixed typo in analyzer

    * fixed tests

    * generalizing binops

    * fixed bool logic binops in analyzer

    * added easier debugging for texprs in sast pretty printer

    * fixed up some tests

    * nested cont printing

    * no s/r error

    * collecitons pass tests

    * whatever for now

    * aight

    * someone fix this so we can actuall just generate an executable

    * need -r flag apparently

    * ugly but works

    * fixing tests

    * removing build dir

    * fixed broken commit by ethan

commit a957c530d6341b2c239c4e7223609fcf87da2617
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 21:35:48 2016 -0500

    updating

commit e2830260197e80cd0d8a4a969070e44fc0a01cf3
Author: Vlad Scherbich <vscherbich@squarespace.com>
```

```
Date:   Sat Dec 17 19:27:54 2016 -0500

    making Actor members private

commit 246fabd66197af595f5b5a5c9272e221cf0a32df
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sat Dec 17 19:22:02 2016 -0500

    Actor prototype

commit 1c68bf5128637f3fc18a11c8db20923e7de98c92
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Sat Dec 17 18:52:58 2016 -0500

    lambdas appear to work (#113)

commit af174e75187de6c85c0a3ec5175922c59070f96f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 17:29:04 2016 -0500

    definign custom error

commit bdf9194ec24cfc4acdacffd17a51129bd5738ef5
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 17:22:32 2016 -0500

    adding scanner errors

commit b205161226fd98b8056b6471be675fde1b67fe7e
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 16:52:30 2016 -0500

    Transpile (#112)

    WE DOING IT BOYS

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing
```

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* approaching

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

* Recursive functions work now

Changed Message_t, Actor_t, Pool_t

* attempt at resolving builtin container functions

* fixing sexpr

* progress on variables

* fixed locals?

* can print all types, just need to bring in analysis

* adding println definitions for more types

* spacing

* variables work for ints chars doubles and bools

* cleanup

* t_expr everywhere

* variables seem to work

* renaming all tests to use Println

* trying to make bools print right

* fixing uop tests because we don't have bitwise negation

* gen pi checks out

* gcd works now

* oops

* fixed typo in analyzer

* fixed tests

* generalizing binops

* fixed bool logic binops in analyzer

* added easier debugging for texprs in sast pretty printer

* fixed up some tests

* nested cont printing

* no s/r error

* collecitons pass tests

* whatever for now

* aight

* someone fix this so we can actuall just generate an executable

```
    * need -r flag apparently

    * ugly but works

    * fixing tests

    * removing build dir

    * strings work, testing is cleaner

    * fixed broken commit by ethan

    * fixed make

commit ccc1203caa92c548b223f5c265f542fc39c3ce99
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 16:18:08 2016 -0500

    removing build dir

commit 2ff3ffe1aa791ee3886593bcb1bb79fa955644ae
Merge: e568adc 831b854
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 16:17:06 2016 -0500

    fixed ethan commit

commit e568adc080ff4586b93cbc5d9b7208ddc1080e12
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 16:14:34 2016 -0500

    fixing tests

commit 831b85470eabe5ebee0807670a0f981b1fad9b3b
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 06:00:55 2016 -0500

    ugly but works

commit 7b42084db857e943b790884bb94ac68820614268
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 17 06:00:43 2016 -0500

    need -r flag apparently

commit 034feb1c7482f68762259961f734c3924334dd8b
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 17 01:44:34 2016 -0500

    someone fix this so we can actuall just generate an executable

commit 74c083f3bb9024e7a55142923cd374a1957bebec
Merge: 0da6011 73c4a6a
Author: Howon <howonbyun@gmail.com>
```

```
Date:   Sat Dec 17 01:15:02 2016 -0500

    comprehensive change to keep everything sane. using corebuild instead

commit 0da6011b5534309b22990fc34f0d607e2285ac7f
Merge: 94957c7 973a932
Author: Howon <howonbyun@gmail.com>
Date:   Fri Dec 16 23:22:30 2016 -0500

    transpile sans actor

commit 73c4a6a430ffe124f5f6df797d2aa710c475bf5d
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Fri Dec 16 22:14:56 2016 -0500

    deleting left-over out with no in test file (#109)

commit 1b76eb83be4d9b1a19c669dab4eb85269b2eb31b
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Fri Dec 16 22:03:18 2016 -0500

    Jh/scoping (#111)

    * program is list of functions and then main

    * trying to do scope properly

    * scoping works minus nested functions

    * oops remove debugging print

commit 52e1200f698a580be5c1bd88451d110ada4baac5
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Fri Dec 16 22:00:20 2016 -0500

    more tests parse (#110)

commit 94957c76975ba5e4e6a4481ea81a49084edba208
Author: Howon <howonbyun@gmail.com>
Date:   Fri Dec 16 11:38:20 2016 -0500

    aight

commit f1d04b24509e1721a2aa95adaed5315a1e087dae
Author: Howon <howonbyun@gmail.com>
Date:   Fri Dec 16 05:37:17 2016 -0500

    whatever for now

commit 973a93263567dc6be604010470dbadc131c0b333
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Thu Dec 15 23:20:16 2016 -0500

    fixed analysis of variable decl when fcall is involved (#108)
```

```
commit c8d4ef214a412dd80ff1a2f837691cd78b1d390d
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Thu Dec 15 20:59:45 2016 -0500

    Jh/firstclassfuncs (#107)

    * started changes

    * changed functions in SAST

    * fixed parser/ast to match sast naming

    * front-end fixxed

    * more analyzer changes

    * small formatting change

    * first class funcs through sast

    * wow I spent too long on this

    * codegen in progress

    * cannot call main

    * more refactor, compiles

    * FUNCTIONS WORK

commit 86b1247fe563f9075580a2b42008a44dde5ce5c5
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Thu Dec 15 05:54:04 2016 -0500

    better syntax errors for parsing (#106)

commit b12bd1f97bb2d51449cbbeed9e25ba165f66798d
Merge: 2f23ccf fee89d4
Author: Howon <howonbyun@gmail.com>
Date:   Thu Dec 15 05:44:12 2016 -0500

    Merge branch 'master' of https://github.com/Howon/Oscar into native

commit 2f23ccfd8bcb1d84cc82b55c83439a10468459b5
Author: Howon <howonbyun@gmail.com>
Date:   Thu Dec 15 05:32:44 2016 -0500

    collecitons pass tests

commit 56e4fa7ca70386481548aedd1abbdbfd2001e433
Merge: a7adbd3 431a444
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 14 22:11:07 2016 -0500
```

```
    merge

commit fee89d4e99d7c70c392a66ec9c4ffbb317af34db
Author: anthonyholley <anthony.j.holley@columbia.edu>
Date:   Wed Dec 14 18:01:03 2016 -0500

    Ah/more tests (#86)

    * better gitignore

    * scan for gen_pi checks out

    * Bool Test

    * Some compiler test files

    * added scanning test for lambda type

    * test files update

    * Added more .out's

    * re-added gitignore; removed else if

    * Corrected tests for doubles, sets and maps

    * Higher precision for double tests

    * Added some tests for mutability of types within actors

    * added tests for recursion, lambda functions and list reversal

    * fixed parsing errors for recursion and list reversal tests

    * added basic test for pools and started adding tests for compiler errors

    * added the vast majority of compiler error test cases from semantic analyzer

    * corrected .out files in errors and changed println to Println

    * Capitalized built-in functions

    * fixed typos

    * added test for nested lambdas

    * this should fix a couple of weird indents

    * we don't need this anymore

    * corrected pool test

    * fixed some char/string outputs by removing quote marks
    * failing tests are now checked properly

    * errors should be written to stderr

    * fixing test

    * prettify

    * removing temp

    * adding newline to gitignore

    * removing temp

commit ab00a26bbf5559c75e8e13e0f376f38d168b734c
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Dec 14 03:45:32 2016 -0500

    Jh/fixbool (#102)

    * fixed typo in analyzer

    * fixed tests

    * generalizing binops

    * fixed bool logic binops in analyzer

    * added easier debugging for texprs in sast pretty printer

    * fixed up some tests

    * fixed reversed order in sast

    * started to make printing bools

    * fixed ifs

    * if/else maybe finally working

    * stdlib!

    * stdlib integration, printing bools worksgs!

    * tests fixed / a few more added

    * added includes to gitignore

commit 431a44424cf9672b0b17e39d422485eda32722fd
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 14 02:29:22 2016 -0500

    Analyzer (#100)
```

* reference manual added

* referting accidental src/ changes

* final LRM

* initial hamt commit

* hamt testing and stuff

* testing started

* c extension test started

* testing testing

* adding variable length printing

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

* Recursive functions work now

Changed Message_t, Actor_t, Pool_t

* attempt at resolving builtin container functions

* fixing sexpr

* progress on variables

* fixed locals?

* can print all types, just need to bring in analysis

* adding println definitions for more types

* spacing

* variables work for ints chars doubles and bools

* cleanup

* t_expr everywhere

* variables seem to work

* renaming all tests to use Println

* trying to make bools print right

* fixing uop tests because we don't have bitwise negation

* gen pi checks out

* gcd works now

* oops

* fixed typo in analyzer

* fixed tests

* generalizing binops

* fixed bool logic binops in analyzer

```
    * added easier debugging for texprs in sast pretty printer

    * fixed up some tests

    * nested cont printing

    * no s/r error

commit 49b0df4b528b9fd97d972e4f4e6ca93372cda0a8
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Dec 14 02:26:20 2016 -0500

    Jh/fixbool (#101)

    * fixed typo in analyzer

    * fixed tests

    * generalizing binops

    * fixed bool logic binops in analyzer

    * added easier debugging for texprs in sast pretty printer

    * fixed up some tests

    * fixed reversed order in sast

    * started to make printing bools

    * fixed ifs

    * if/else maybe finally working

commit a7adbd3f00a470b4dcab4850492ba2bf70a200d9
Merge: 72c7ab4 2dae61d
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 14 02:00:08 2016 -0500

    merging from analyzer

commit 2dae61dbba49f0326ad49e0f2d1adaea50959395
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 14 01:32:43 2016 -0500

    no s/r error

commit 36d3f6dc60b83380f39b0775cc5716f50032a835
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 14 01:24:20 2016 -0500

    nested cont printing
```

```
commit d4f8ba9a0e9c720da013076e3b1b279a553d18e8
Merge: 136a578 5aeeab2
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 13 22:43:38 2016 -0500

    Merge remote-tracking branch 'origin/jh/fixbool' into analyzer

commit 5aeeab252fd98ab019eee2da123ee142d5c1a773
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 20:50:15 2016 -0500

    fixed up some tests

commit ea38b2e26b42891e7cab02e2b436f22e50cfa16c
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 20:19:55 2016 -0500

    added easier debugging for texprs in sast pretty printer

commit 03227c60734e4097e67d5c5b7c279a0c25f96e96
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 20:08:57 2016 -0500

    fixed bool logic binops in analyzer

commit e92f4e1498c01ab8d7483df17f4506bf92af9168
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 20:01:32 2016 -0500

    generalizing binops

commit 8d5628983ae2f373c312290b24e249d4a7da374c
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 18:30:37 2016 -0500

    fixed tests

commit 4345a75047d6a0afc95c049d1a5cfb816b407470
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Dec 13 18:18:05 2016 -0500

    fixed typo in analyzer

commit 136a5789c99749778030e5f41bd3732c78fa4dba
Merge: 190932b 1565048
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 13 18:17:43 2016 -0500

    Merge branch 'master' of https://github.com/Howon/Oscar into analyzer

commit 190932bcf78c6d7c68e0f4b68ed04f82f394d0aa
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 13 18:17:36 2016 -0500
```

```
    oops

commit 1565048da25451723fe3248ece13e05037ec43b5
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Tue Dec 13 00:35:30 2016 -0500

    Codegen/sast integretion (#94)

    * progress on variables

    * fixed locals?

    * can print all types, just need to bring in analysis

    * adding println definitions for more types

    * spacing

    * variables work for ints chars doubles and bools

    * cleanup

    * variables seem to work

    * renaming all tests to use Println

    * trying to make bools print right

    * fixing uop tests because we don't have bitwise negation

    * adding newlines for tests

commit 297eba4a87b3f979952de7975213c29048952447
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Tue Dec 13 00:21:43 2016 -0500

    Analyzer (#96)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing
```

* adding variable length printing

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

* Recursive functions work now

Changed Message_t, Actor_t, Pool_t

* attempt at resolving builtin container functions

* fixing sexpr

```
    * progress on variables

    * fixed locals?

    * can print all types, just need to bring in analysis

    * adding println definitions for more types

    * spacing

    * variables work for ints chars doubles and bools

    * cleanup

    * t_expr everywhere

    * variables seem to work

    * renaming all tests to use Println

    * trying to make bools print right

    * fixing uop tests because we don't have bitwise negation

    * gen pi checks out

    * gcd works now

commit 045bab8864c1c42d42266f10ff2e9874fbccead6
Merge: 55f2ce5 e65b0db
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 12 22:42:03 2016 -0500

    merge

commit 55f2ce5dbb1e7eb588093e47933d36ea534b9bfb
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 12 22:41:12 2016 -0500

    gcd works now

commit e65b0dba31abed7d753cedefff42a9b543f1f00c
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Sun Dec 11 16:36:45 2016 -0500

    Analyzer (#95)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit
```

* hamt testing and stuff

* testing started

* c extension test started

* testing testing

* adding variable length printing

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

```
    * Recursive functions work now

    Changed Message_t, Actor_t, Pool_t

    * attempt at resolving builtin container functions

    * fixing sexpr

    * progress on variables

    * fixed locals?

    * can print all types, just need to bring in analysis

    * adding println definitions for more types

    * spacing

    * variables work for ints chars doubles and bools

    * cleanup

    * t_expr everywhere

    * variables seem to work

    * renaming all tests to use Println

    * trying to make bools print right

    * fixing uop tests because we don't have bitwise negation

    * gen pi checks out

commit 5fd1d948739be1bbd440de3c6654daacfe75fa77
Merge: 4d6b84a f60e083
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 11 16:35:05 2016 -0500

    master ready

commit 4d6b84a492b6f62ebf169d531399435b0e69c5d8
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 10 22:14:19 2016 -0500

    gen pi checks out

commit f60e0831c3fc0f283f16bde947f38e53b1fa3aac
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 10 21:31:06 2016 -0500

    fixing uop tests because we don't have bitwise negation
```

```
commit b444a789c140061575da84e10fd43f2d3b8d0fd1
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 10 21:30:51 2016 -0500

    trying to make bools print right

commit bb02468549560bc4329eb5d5b9928ac72b7ce8a0
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 10 21:09:13 2016 -0500

    renaming all tests to use Println

commit 15ada5726cbb4e9be95df585bfc781bf212b6714
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 10 21:08:51 2016 -0500

    variables seem to work

commit cb1e3c697572572499d33dfd70f580a266578a5d
Merge: e63d52e edd8715
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Dec 10 17:34:20 2016 -0500

    getting updated changes from master

commit edd8715379f550dc2b6e768943f650150d93e39d
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 10 17:26:20 2016 -0500

    T expr master race (#93)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing

    * default adds a newline to print

    * adding printing of binops

    * adding mod
    * renamed n stuff

    * updating

    * making testing testy

    * removed unneeded junks

    * .out files added

    * adding newlines to end of test files

    * damn it jibb

    * val decl separated

    * analyzer started

    * anal

    * for now

    * here jibb

    * almost done

    * fuck you howon

    * analyzer funcdecl check done

    * shift reduce error for some reason

    * Lambda should be in function scope now

    * analyzer works. need to add more builtins

    * need to figure out builtins

    * Recursive functions work now

    Changed Message_t, Actor_t, Pool_t

    * attempt at resolving builtin container functions

    * fixing sexpr

    * t_expr everywhere

commit 2a3082d3e6d437d2511bda5af218692b498e897b
Merge: fb9d8d5 605147f
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 10 17:21:08 2016 -0500
```

```
    with master

commit fb9d8d5195208b2c14948df6b93f9b65bd7faa2e
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 10 17:14:49 2016 -0500

    t_expr everywhere

commit e63d52e5d29e31680fe4e93e2d0beb8a468b3ebe
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 22:32:57 2016 -0500

    cleanup

commit 0f6e4104faeaa9ab4326dda9b2ee9e009caee6ed
Merge: 1872ebd 605147f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 22:30:26 2016 -0500

    merging conditionals from master

commit 605147f3d9981527badefd67f072f50cac813882
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Wed Dec 7 22:19:33 2016 -0500

    Vs/codegen (#92)

    * rebased on master

    * added if/else branches into codegen + fixed up some func return types in tests

    * removed extra String_Lit from codegen expr builder

commit 1872ebd249e5d35c951f15ed6052625ec4ba5bab
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 22:17:04 2016 -0500

    variables work for ints chars doubles and bools

commit 60e323a321d23251a91ff321348db6fa9b8ab93b
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 21:53:25 2016 -0500

    spacing

commit faacf59ba1f7bdfb11e093e532b1b2ea4c79c1be
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 21:53:11 2016 -0500

    adding println definitions for more types

commit aceee4482f8c0a638a531856edd912cf6e2eb372
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 21:52:39 2016 -0500
```

```
    can print all types, just need to bring in analysis

commit 919faee998a125fbdf41a9287fc582ebfb334f12
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 21:40:06 2016 -0500

    fixed locals?

commit 3dae602562928f8c395c413476d2518a719d1112
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Dec 7 21:24:36 2016 -0500

    progress on variables

commit a681c6c1cfa5bb9d12c4bd361e93b38ffeb11112
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 19:11:08 2016 -0500

    Analyzer (#90)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing

    * default adds a newline to print

    * adding printing of binops

    * adding mod

    * renamed n stuff

    * updating

    * making testing testy

    * removed unneeded junks

    * .out files added
    * adding newlines to end of test files

    * damn it jibb

    * val decl separated

    * analyzer started

    * anal

    * for now

    * here jibb

    * almost done

    * fuck you howon

    * analyzer funcdecl check done

    * shift reduce error for some reason

    * Lambda should be in function scope now

    * analyzer works. need to add more builtins

    * need to figure out builtins

    * Recursive functions work now

    Changed Message_t, Actor_t, Pool_t

    * attempt at resolving builtin container functions

    * fixing sexpr

commit eee015694d8059de1968c6c8f9cc367a9bb3efad
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 18:22:57 2016 -0500

    fixing sexpr

commit 6d26c1499ae82e8bb2c2587030c693f90c50af3d
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Dec 7 15:36:20 2016 -0500

    tests work again (#89)

commit ecd4ebedf146ae9d1a026d8649069ee7ca1c1061
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Dec 7 15:08:45 2016 -0500

    Codegen/sexprs (#88)
```

* reference manual added

* referting accidental src/ changes

* final LRM

* initial hamt commit

* hamt testing and stuff

* testing started

* c extension test started

* testing testing

* adding variable length printing

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* added exclamation to HelloWorld test; started fixing other tests;

* fightin' the good fight

* rebased + simple func call test

* for now

* bool_lits and char_lits now being codegend

```
    * fixed test for funcCall()

    * added UOps code to codegen + tests

    * adding tests that didnt get added

    * adding func formals test

    * improve funcFormals test to only check passing of formals

    * implemented fxn formals and fixed test output

    * removing add

    * updating a bunch of tests

    * adding doubles

    * added immutable var tests

    * making gitignore better

    * more test files added

    * added BITWISE ops to codegen + modified tests

    * added BITWISE ops to codegen + modified tests

    * stuff pulled from howon

    * more stuff resolved

commit 87b3003fe5513347281618a657946a3b500b5673
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 14:51:58 2016 -0500

    Analyzer (#77)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing
```

* adding variable length printing

* default adds a newline to print

* adding printing of binops

* adding mod

* renamed n stuff

* updating

* making testing testy

* removed unneeded junks

* .out files added

* adding newlines to end of test files

* damn it jibb

* val decl separated

* analyzer started

* anal

* for now

* here jibb

* almost done

* fuck you howon

* analyzer funcdecl check done

* shift reduce error for some reason

* Lambda should be in function scope now

* analyzer works. need to add more builtins

* need to figure out builtins

* Recursive functions work now

Changed Message_t, Actor_t, Pool_t

* Jh/fix sast (#87)

* small ast change

```
    * pretty print works

commit 879629b8dab1d9ad1ad6be04336d7dd80e0c6a2a
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 12:14:08 2016 -0500

    attempt at resolving builtin container functions

commit 13bbf6ee083ee576093ce1f838bd49af3c90cb02
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 11:05:11 2016 -0500

    Recursive functions work now

    Changed Message_t, Actor_t, Pool_t

commit ca379c80e7368b88888799e698096209e6da659b
Merge: 3f07bf2 96fa02e
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 03:14:01 2016 -0500

    Merge branch 'master' of https://github.com/Howon/Oscar into analyzer

commit 3f07bf2151ee7336926314e0d9f0c7f533356dbe
Author: Howon <howonbyun@gmail.com>
Date:   Wed Dec 7 03:13:49 2016 -0500

    need to figure out builtins

commit 96fa02ec2084e96b53de2d4cd170da9eabb5e3e4
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Tue Dec 6 20:42:49 2016 -0500

    scanner now actually handles chars correctly (#85)

    * scanner now actually handles chars correctly

    * more ignore

commit 938fcc551e1c44166c0ed532e4d1fef85a5fc95e
Merge: c917837 ad88461
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 6 04:09:14 2016 -0500

    analyzer done. need to add more builtin funcs

commit ad88461a0bfa9d0c2af62e9cad02e780eefce827
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 6 04:05:36 2016 -0500

    analyzer works. need to add more builtins

commit d3343bb8235e4b638440a0cab6e820eccfd941ec
```

```
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 6 03:05:07 2016 -0500

    Lambda should be in function scope now

commit c9178376ce477e18f8b5b3374f5b5b5d713e471a
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 6 03:05:07 2016 -0500

    shift reduce error for some reason

commit fb1a5bf8c3b5ae5fa56abc55b31e9a84b282bbb1
Merge: 288dcb5 adec4ac
Author: Howon <howonbyun@gmail.com>
Date:   Tue Dec 6 00:52:19 2016 -0500

    for codegen

commit 288dcb54ce01a3c319ec5debe95b57020166479a
Merge: 0fea2a6 b1df476
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 5 19:52:00 2016 -0500

    analyzer compiles.

commit 0fea2a6eced59833a6be3eaef450db04383ec1f6
Author: Howon <howonbyun@gmail.com>
Date:   Mon Dec 5 14:45:14 2016 -0500

    analyzer funcdecl check done

commit b1df476f674833bdb1a97472526c415146bf93a6
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Mon Dec 5 14:40:27 2016 -0500

    Actually working parsing/pretty printer (#82)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing
    * default adds a newline to print

    * adding printing of binops

    * adding mod

    * renamed n stuff

    * updating

    * making testing testy

    * removed unneeded junks

    * .out files added

    * adding newlines to end of test files

    * damn it jibb

    * val decl separated

    * analyzer started

    * anal

    * for now

    * here jibb

    * almost done

    * fuck you howon

    * parser/ast/prettyprint/scanner

    * can now make oscar, prettyprint moved to oscar

    * fixed testing call for scanner

    * more consistent code style

commit adec4ac0b34596b17f325944343089c99f580d92
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sun Dec 4 18:26:08 2016 -0500

    fuck you howon

commit d49b5a99325248962aaba0ed096cdba8464c85ae
Author: Howon <howonbyun@gmail.com>
Date:   Sun Dec 4 04:53:07 2016 -0500

    almost done
```

```
commit 14d32c01792c59f43aa3cf56dd5dacc607edc97c
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 3 16:44:33 2016 -0500

    here jibb

commit c28c5e6de49f7a9f96a50f4552db5ad4e1ccc2ba
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 3 13:05:02 2016 -0500

    for now

commit aa3f3cc85830472330a205022f404a409e95a517
Merge: b1593d8 2441e36
Author: Howon <howonbyun@gmail.com>
Date:   Sat Dec 3 03:24:52 2016 -0500

    kill me

commit b1593d8a287d7d3a4b5ebe531b8757da469ced0d
Author: Howon <howonbyun@gmail.com>
Date:   Fri Dec 2 21:18:50 2016 -0500

    anal

commit 2441e367c0b608d42b99d932f30d463d06283637
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Fri Dec 2 21:14:50 2016 -0500

    fixed Makefile after the prettyprinter additon (#76)

commit 6737447b264eb1f0ba0fce2b220398d03ecbbfbb
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Thu Dec 1 23:07:27 2016 -0500

    Ah/testing (#74)

    * some of prettyprint

    * some of prettyprint

    * pretty printer should be working

    * pretty printer update

    * some of prettyprint

    * pretty printer should be working

    * pretty printer update

    * removed field from ast
    * pretty printing!

    * pretty printer in Makefile

    * removed some unnecessary new lines

    * fixed typo in test

    * I just want this makefile to work :(

    * fixed uop, spacing

commit 469316d4ebef2c49f283a09ca34324cec07c7c02
Author: Howon <howonbyun@gmail.com>
Date:   Wed Nov 30 22:46:53 2016 -0500

    analyzer started

commit 77c6bf3e941efa321d2323d9073d34bede3c2642
Author: Howon <howonbyun@gmail.com>
Date:   Wed Nov 30 19:24:47 2016 -0500

    val decl separated

commit 27f70b669477d8ee8dcb2214e17265565560ff2c
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 21:28:44 2016 -0500

    Testing (#62)

    * reference manual added

    * referting accidental src/ changes

    * final LRM

    * initial hamt commit

    * hamt testing and stuff

    * testing started

    * c extension test started

    * testing testing

    * adding variable length printing

    * default adds a newline to print

    * adding printing of binops

    * adding mod
    * renamed n stuff

    * updating

    * making testing testy

    * removed unneeded junks

    * .out files added

    * adding newlines to end of test files

    * damn it jibb

commit dfab6e3d659f7018107ef54c5159f1dd6bb8ed15
Merge: f43075f f833bdc
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 19:19:50 2016 -0500

    Merge branch 'master' of https://github.com/Howon/Oscar into analyzer

commit f43075f4791ab48f66fb35f499b2bd641c60dcf2
Merge: 3fe0907 fbc477e
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 18:22:43 2016 -0500

    lets merge

commit 3fe09074f9d8acdca3d4ae3931f677e88dc7ff47
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 18:21:08 2016 -0500

    damn it jibb

commit fbc477e2272660ec7c5e071a9b30c2ef59bb4c48
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Tue Nov 29 15:00:14 2016 -0500

    adding newlines to end of test files

commit 4f9ca09354a770f5b05fe901e4be740124b9b742
Merge: 8f24130 eae525f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Tue Nov 29 15:00:02 2016 -0500

    updating codegen to match more_codegen

commit 8f24130094b428a074ea320193ec9b6fc40d4e3f
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 02:38:59 2016 -0500

    .out files added

commit c56ca1c2369e1fab534ef66dc55cc98c1aa35a08
```

```
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 02:07:48 2016 -0500

    removed unneeded junks

commit 72c7ab44fe19cecd7df80eddeb57bfac398212da
Author: Howon <howonbyun@gmail.com>
Date:   Tue Nov 29 00:53:53 2016 -0500

    approaching

commit 740563cc7ae88b7af96e3563f8a888e86dedeef7
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Thu Nov 24 00:08:28 2016 -0500

    making testing testy

commit eae525f2c4a7b3133e866ee6f8bb280ce0a6f6fd
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 18:25:15 2016 -0500

    updating

commit 2ed602ebaa21dc4b0e1be86555f8f94af1221292
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 18:23:45 2016 -0500

    renamed n stuff

commit f833bdc1ba502802c276dfbf84b6b2ad7a8a7da0
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Sat Nov 19 18:15:37 2016 -0500

    removing some commented blocks (#68)

commit 306467c382be8e5336e54b5d361bb0d719c3a894
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Sat Nov 19 17:54:47 2016 -0500

    Fixed issues with AST / Parser (#65)

    * fixed mutables (#59)

    * Fixed 'chr' RegEx + added a space to it (#60)

    * Jh/pasta tea fix (#61)

    * fixed mutables

    * Make clean is now cleaner

    * fixed vdecl

    * Jh/pasta tea fix (#63)
```

* fixed mutables

* Make clean is now cleaner

* fixed vdecl

* fixed statement lists in parser

* fixed cond / loops

* fixed statement lists in parser

* fixed cond / loops

* Jh/pasta tea fix (#64)

* fixed mutables

* Make clean is now cleaner

* fixed vdecl

* fixed statement lists in parser

* fixed cond / loops

* fixed statement lists in parser

* fixed cond / loops

* fixed lambda mismatch

* removed todo

* fixed cont_lits, added actions for actor/pool

* fixed lambda mismatch

* removed todo

* fixed cont_lits, added actions for actor/pool

* removed some TODOs

* fixed mutables (#59)

* Jh/pasta tea fix (#61)

* fixed mutables

* Make clean is now cleaner

* fixed vdecl

```
    * Jh/pasta tea fix (#63)

    * fixed mutables

    * Make clean is now cleaner

    * fixed vdecl

    * fixed statement lists in parser

    * fixed cond / loops

    * fixed statement lists in parser

    * fixed cond / loops

    * Jh/pasta tea fix (#64)

    * fixed mutables

    * Make clean is now cleaner

    * fixed vdecl

    * fixed statement lists in parser

    * fixed cond / loops

    * fixed statement lists in parser

    * fixed cond / loops

    * fixed lambda mismatch

    * removed todo

    * fixed cont_lits, added actions for actor/pool

    * fixed lambda mismatch

    * removed todo

    * fixed cont_lits, added actions for actor/pool

    * removed some TODOs

commit 37263d63e7d9896175cd7c0b26ee716beb4e5cf7
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 17:38:50 2016 -0500

    adding mod

commit 4d3f71dc1ceaf1439db254511e4e72616d7e8311
```

```
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 17:35:55 2016 -0500

    adding printing of binops

commit ad6b4236ba434e0852324010073c9ace57f687a3
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 17:12:04 2016 -0500

    default adds a newline to print

commit f8293ee20ef369caa411f76f020443b8451fb428
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 19 17:02:41 2016 -0500

    adding variable length printing

commit 446605bbb446f3fbbf06bb5cd2af70bc1560596d
Merge: f260b8a f57f9fd
Author: Howon <howonbyun@gmail.com>
Date:   Sat Nov 19 16:37:27 2016 -0500

    merging from master

commit 0b3d9164d3c85c1bbc58d023f10525fc44a9c3f1
Merge: b37fc34 f57f9fd
Author: Howon <howonbyun@gmail.com>
Date:   Sat Nov 19 16:35:22 2016 -0500

    merge fix

commit b37fc34be8ce12e3f65e4dddeca289936e8332ec
Author: Howon <howonbyun@gmail.com>
Date:   Sat Nov 19 16:31:06 2016 -0500

    testing testing

commit f57f9fd09a9fdd2eacd569db8554311581751464
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Sat Nov 19 15:42:46 2016 -0500

    Fixed 'chr' RegEx + added a space to it (#58)

commit 97c3b502baa1382c97a8acd92c08eb1991dfbd94
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sat Nov 19 14:02:53 2016 -0500

    Helloworld (#56)

    * init

    * updating hello_world with menhir's results

    * moving codegen
    * first pass at creating print fxn

    * removed all the code we don't need

    * making makefile makey

    * update codgen

    * codegen and makefile actually working

    * ignoring cmx and .o

    * renaming ast.mli to ast.ml

    * EVERYTHING COMPILES

    * HELLO WORLD

    * final changes

    * making vlad happy

    * making vlad happy

    * removing hello world txt

commit 7984a273ee8d76d5b7802eb6e7085b36a740f57f
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Thu Nov 17 02:26:35 2016 -0500

    Jh/fix code issues (#55)

    * fixed scanner stuff

    * fixed parser

    * fixed typos in parser

    * fixed ast

    * added unit, fixed more stuff in ast and parser

    * fixed parser typo

    * renamed ast

    * reverted type changes

    * deleted the crap

commit f260b8af14e2120ee1463f82cb30c9bfec6a9eb8
Author: Howon <howonbyun@gmail.com>
Date:   Wed Nov 16 19:41:37 2016 -0500
```

```
    c extension test started

commit 13b3612d219d7bc1409196393210ecb8e22575b5
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Nov 16 16:50:16 2016 -0500

    made sast to match ast (#49)

commit 23fed1a27707cd7198309579f7b001b789a452e7
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Tue Nov 15 22:52:05 2016 -0500

    Pasta/vassign (#45)

    * refactored parser, 2 s/r conflicts

    * added arrow, renamed LANGLE

    * fixed loops, fixed shift/reduce, fixed a lot of stuff

    * fixed issue with merge

    * added bitwise ops back

    * fixed indentation

commit 153c42438e1bd73cb03c34c59b369a7710dd7565
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Tue Nov 15 22:51:40 2016 -0500

    fixing scanner, it compiles (#47)

commit a75bdcacdec00a8e2cde0592b7da342999d94ff2
Author: Howon <howonbyun@gmail.com>
Date:   Sun Nov 13 03:09:50 2016 -0500

    testing started

commit 6760ae80e832cb08a71b9bc07c60712c88626e63
Merge: 764bf8a 485d3c9
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Mon Nov 14 11:44:58 2016 -0500

    Merge pull request #46 from Howon/fix_ast

    fix up AST after @jibben refactor

commit 485d3c906ae3a4cb71656a3a2494ac0bb07b43dd
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Nov 13 16:18:15 2016 -0500

    fix up AST after @jibben refactor
```

```
commit 764bf8a39858f5bb65bf2b8b94dd6cbed9726b84
Merge: 7cbf6d8 2be6483
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sat Nov 12 18:12:48 2016 -0500

    Merge pull request #42 from Howon/pasta/goodbye_dot

    goodbye dot

commit 2be6483ef0338b19f3df8320d9deec839fcbb157
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 18:09:20 2016 -0500

    goodbye dot

commit 7cbf6d83f305fefbc0e2581f42e459b4ccf2b510
Merge: e0b7b55 6a0a5d2
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sat Nov 12 18:05:05 2016 -0500

    Merge pull request #40 from Howon/pasta/fix_bitwise_ops

    making bitwise_not a unop

commit 6a0a5d2a0f3f7f40b1b91a137e6055378530aabb
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 18:00:28 2016 -0500

    fixing it once again cause i forgot to put the bitwise ops in bit ops

commit 3d0248c7b294e4c858a9343bd92edc4e67646a50
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:54:28 2016 -0500

    updating uops

commit 832fb07e05b5a2d4e4619b4dbdf209f42fd0fab7
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:49:00 2016 -0500

    getting rid of bit_op becasue bin_op is all we need

commit 9884be2eb1b0a550a26ebc7e591f483e63d572ef
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:40:21 2016 -0500

    making bitwise_not a unop

commit e0b7b551aaae414aa25cd283cff54ff976e50637
Merge: ff761d2 d0728dc
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sat Nov 12 17:23:58 2016 -0500

    Merge pull request #39 from Howon/pasta/add_bitwise_ops
    STRAIT MERGEIN

commit d0728dc271ad08e81ccd2dff0b48ced68f6a1969
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:17:43 2016 -0500

    updating parse tests to use a bitop

commit f70f178e7cc4cb45ec75fb042e47b0fcfa0dad0d
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:17:26 2016 -0500

    adding bitops to parser

commit 81b605863c58e3c9bfb20765cecc5cfed522f44e
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 17:17:14 2016 -0500

    reording stuff

commit ff761d2d51b1fabd3d9dd6fc3e7701c42946392c
Merge: 3019e1c 03556aa
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 16:45:43 2016 -0500

    merging

commit 3019e1c4f7d77fa039ec2dc41fe860599d7e76d4
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Nov 12 15:10:18 2016 -0500

    lambdas now lambda

commit 03556aaf9d864852c8a4e9c0118ce28352e41344
Merge: 2bfc3b3 4c483b4
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Sat Nov 12 14:26:57 2016 -0500

    Merge pull request #35 from Howon/pasta/if_sr_fix

    fixed shift/reduce with if-else

commit 4c483b45e2abb897884ed63489821e8ef21d72d9
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Sat Nov 12 14:17:11 2016 -0500

    fixed shift/reduce with if-else

commit e9d50ff6ec1a085cc09781f861d42105358591fc
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Nov 9 21:54:31 2016 -0500

    header
```

```
commit 3e1c36ce44efdb03b285f499ddc291bff2cc3a58
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Nov 9 21:54:01 2016 -0500

    adding parse tests for use with menhir

commit b2996ac2fbc681cbdea99fddcad93dfad57a80dc
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Nov 9 21:53:41 2016 -0500

    lambas have to be surrounded in parens....

commit ba5265354b57b784a9e9590b24cbe1b4ac899f1d
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Nov 9 19:45:52 2016 -0500

    adding lambda functs

commit f6c8ce69089c8e185317dad0229fea4ab4a214f9
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Nov 9 19:08:53 2016 -0500

    pattern matching is now good to go

commit 2bfc3b38fff1d05a8e38b43c62818c0ad5e042ef
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Mon Nov 7 03:20:43 2016 -0500

    fixed some 'never reduced' errors and something else (#32)

commit ef201d3b3f3e627e5f0aee4d18673c2bd91b9892
Author: Howon <howonbyun@gmail.com>
Date:   Mon Nov 7 03:15:28 2016 -0500

    hamt testing and stuff

commit 0bf63b30c8bfa4b5d5ecb299f9e2aeaecb1078a6
Author: Howon <howonbyun@gmail.com>
Date:   Mon Nov 7 02:58:09 2016 -0500

    initial hamt commit

commit 86591474d8b5f1606048803abefa7b1abceb628e
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Nov 6 18:12:44 2016 -0500

    at top level, now actors and functions have the right actions. but we still need to make sure that pattern
matching and lambdas work

commit 3c10737879b79641481fcca956006f6c67f7bcf0
Merge: 101dbd2 b88409b
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Sun Nov 6 17:54:13 2016 -0500
```

```
    Merge pull request #16 from Howon/vs/scanner_parser

    Vs/scanner parser

commit b88409b1e2cc011e3d656993609429f913c5b25b
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Nov 6 17:51:25 2016 -0500

    addressed @jibben comment

commit 101dbd2f182a91233b1c5536961584677543305e
Merge: 3143204 1166552
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sun Nov 6 16:35:42 2016 -0500

    Merge pull request #18 from Howon/pasta/build_lists

    build up lists correctly

commit 1166552e38a68136e5164c90106a1b27c951d63d
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Nov 6 16:33:40 2016 -0500

    fixing whitespace

commit f427e460c9202c2b7862611db68ca8b05323f74c
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sun Nov 6 16:26:52 2016 -0500

    building lists up correctly. enforces program order of messages, actors, functions

commit 681ebd815836bee928bc8c8e8439d663cdf11d44
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Tue Nov 1 08:27:04 2016 -0400

    a few op name fixes in parser; small fixes in scanner

commit 3143204ceb56e51c2973a3a343d92277e80460f0
Merge: 19cdbdc 60c447a
Author: anthonyholley <anthony.j.holley@columbia.edu>
Date:   Sun Nov 6 14:46:16 2016 -0500

    Merge pull request #17 from Howon/ah/optional

    fixing optional type

commit 60c447ae8139787d07c12cf5452855f5d8e3629e
Author: Anthony Holley <ajh2186@columbia.edu>
Date:   Sun Nov 6 14:34:44 2016 -0500

    fixing optional type

commit 19cdbdca65c0f4e6f81dcb99e7b97d115e678ffb
```

```
Merge: 403d791 2920716
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sun Nov 6 13:33:33 2016 -0500

    Merge pull request #13 from Howon/vs/ast

    ast: brand new day with a brand new branch

commit 29207163ec65152ff40e0be8889cb80025058803
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Nov 6 13:32:35 2016 -0500

    moving all 'oscar.ml' content into 'ast.mli'

commit 1ec2d0d2a60a061929bf6fa7a07d1f3957c9089e
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Tue Nov 1 08:39:15 2016 -0400

    remove parser from PR

commit e7c4eae5017163361ead4b21bd2ffadcaffced60
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Tue Nov 1 08:36:43 2016 -0400

    updated some op types in parer; started working on oscar.ml

commit e761d94574e35f225305f22974800c370d1352af
Author: Vlad Scherbich <vscherbich@squarespace.com>
Date:   Sun Oct 30 11:44:25 2016 -0400

    ast: brand new day with a brand new branch

commit b2b159d5442db0eecf491456e6060d380d376185
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Wed Oct 19 00:13:44 2016 -0400

    added all tokens to parser

commit 403d7913123b0375bcdfa1c23e0969e2c13bba3d
Merge: 34b68a4 75388be
Author: xvladus1 <vlad.scherbich@gmail.com>
Date:   Mon Oct 31 22:25:50 2016 -0400

    Merge pull request #15 from Howon/bug/scanner_char

    fixed typo in scanner

commit 34b68a4da632626f91d591ebe408d90ca7ffd226
Author: Howon <howonbyun@gmail.com>
Date:   Mon Oct 31 13:52:05 2016 -0700

    Doc (#2)

    * reference manual added
    * referting accidental src/ changes

    * final LRM

commit 2d95bca31410d117c3754dc024b7c72f988247af
Author: Howon <howonbyun@gmail.com>
Date:   Mon Oct 31 16:49:18 2016 -0400

    final LRM

commit 75388beacbad4dff5a20e876d9c02ba62f1a398c
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Mon Oct 31 16:09:41 2016 -0400

    fixed typo in scanner

commit 65c2fa3a99b6102c827d4080272b6e17c9c73bd1
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Mon Oct 31 14:34:17 2016 -0400

    Jh/parser (#14)

    * added most tokens to parser

    * added all tokens to parser

    * added bitwise xor

    * added most tokens to parser

    * added all tokens to parser

    * added bitwise xor

    * added all of ethan's changes

    * fixed for style guidelines

    * fixed ast for style

    * jk now scanner is pretty

    * removed let, renamed some tokens

    * did more parser work, see TODOs

    * added stuff to .gitignore

    * more work on parser/scanner

    * fixed function token names

    * fix merge conflict on '.gitignore'; standardized '<' and '>' token names across scanner and parser
```

```
commit f31621e44d1fae9ebb2ed41762f58dd5240ccb4c
Merge: bb9d215 5cc00f6
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Fri Oct 28 17:06:50 2016 -0400

    Merge pull request #3 from Howon/Howon-patch-2

    MERGE IT

commit bb9d215f0378dc58626100e4be6d0a0008758fa5
Merge: 084583d dd8e41f
Author: Howon <howonbyun@gmail.com>
Date:   Sat Oct 22 19:52:00 2016 -0400

    Merge pull request #9 from Howon/scanner

    Scanner

commit dd8e41f288c95d6daa43bcd6c0d3df8b5203e26f
Merge: 9223938 6772d95
Author: Howon <howonbyun@gmail.com>
Date:   Sat Oct 22 19:43:47 2016 -0400

    Merge pull request #7 from Howon/hb/scanner

    Hb/scanner

commit 6772d95419c1f9cdff209e6d794baf0dfe32de47
Author: Howon <howonbyun@gmail.com>
Date:   Sat Oct 22 19:23:43 2016 -0400

    consistent name space

commit 084583d7e7a167193bdf222d4b499c530b817b9c
Merge: 81b5ce6 5f39d0c
Author: Howon <howonbyun@gmail.com>
Date:   Sat Oct 22 19:26:46 2016 -0400

    Merge pull request #6 from Howon/revert-1-jh/full-scanner

    Revert "Jh/full scanner"

commit 5f39d0c67178cace5a4963c27e8e736be0cc1118
Author: Howon <howonbyun@gmail.com>
Date:   Sat Oct 22 19:25:18 2016 -0400

    Revert "Jh/full scanner"

commit 81b5ce65e5e7d44a26b464a9e090314ba7ba6e80
Merge: 9223938 086e913
Author: Ethan Adams <aethos@users.noreply.github.com>
Date:   Sat Oct 22 19:18:41 2016 -0400
```

```
    Merge pull request #1 from Howon/jh/full-scanner

    Jh/full scanner

commit abf864a37a942cb76aa7bbb27c4c258e253e0b6e
Author: Howon <howonbyun@gmail.com>
Date:   Fri Oct 21 03:13:57 2016 -0400

    consistent naming convention

commit 086e9131d594ed597b364867b914647ba74ca123
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Fri Oct 21 00:07:00 2016 -0400

    adding receive

commit 134d1709a67b710b311502d6c11c3682384c68c2
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Thu Oct 20 15:17:52 2016 -0400

    Revert "adding colon"

    This reverts commit 2af85cf7a347ce767106ec2fb3a912657017430d.

commit 2af85cf7a347ce767106ec2fb3a912657017430d
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Oct 19 22:09:11 2016 -0400

    adding colon

commit c4d613efd3a52cc31c9a8f8198afcde21232ed4f
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Wed Oct 19 18:49:15 2016 -0400

    adding let, fixing typos

commit 5cc00f6bea3445b0b09c919e8d2aa2ab3de33889
Author: Howon <howonbyun@gmail.com>
Date:   Wed Oct 19 17:19:34 2016 -0400

    Update .gitignore

commit ce59bfac31b18d3851e9de73611c8aa8402023cd
Author: Howon <howonbyun@gmail.com>
Date:   Wed Oct 19 17:15:35 2016 -0400

    referting accidental src/ changes

commit d118f157d2704edf244a47eb880f3c0e34efa022
Merge: 91c0df4 45a40ce
Author: Howon <howonbyun@gmail.com>
Date:   Wed Oct 19 17:15:09 2016 -0400

    Merge branch 'doc' of https://github.com/Howon/Oscar into doc
```

```
commit 91c0df4bfd11849327be93f1bd16ea731e6479c3
Author: Howon <howonbyun@gmail.com>
Date:   Wed Oct 19 17:14:57 2016 -0400

    reference manual added

commit 45a40ce78b18dfa958663e57b07aa39ffe1a49d8
Author: Howon <howonbyun@gmail.com>
Date:   Wed Oct 19 17:14:01 2016 -0400

    reference manual in plain text for convenience

commit 3b387ffab322d82990c1272d63722ec7b1e5f163
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Wed Oct 19 01:38:37 2016 -0400

    lowercase maybe, fixed typo

commit 45f2ec140e68a677fb5bf3897f001c31a7edb624
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Wed Oct 19 00:19:35 2016 -0400

    added bitwise xor

commit b65a6ce2f90a3674c46dd78c6ed1ceefc4580fb3
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Wed Oct 19 00:14:22 2016 -0400

    completed scanner

commit 67e65217d2263f2aec4cc2265f071a644b1212d5
Author: Anthony Holley <ajh2186@columbia.edu>
Date:   Tue Oct 18 21:59:25 2016 -0400

    Added symbols

commit edbb13c9ebd87789b12a36b7c4b0473cab74a5ec
Author: Jibben Hillen <jlh2128@columbia.edu>
Date:   Tue Oct 18 21:07:44 2016 -0400

    added a few new tokens

commit 922393810f3f714b0b97a9426fc796f77a1624d2
Author: Ethan Adams <ea2678@columbia.edu>
Date:   Sat Oct 15 15:09:41 2016 -0400

    adding gitignore and edits

commit 9ec93b256d3432f8cd0b498ca440200c57ffc441
Author: Howon <howonbyun@gmail.com>
Date:   Mon Oct 10 13:17:17 2016 -0400

    files copied from hw1
```

```
commit b02e6dc7ae71703370a43924536f1b849782a1e3
Author: Jibben Hillen <jlh2218@columbia.edu>
Date:   Wed Sep 28 17:00:30 2016 -0400

    readme

commit 7fb40aac440f59e7a7390fe285aee0c8db802fc6
Author: Howon <howonbyun@gmail.com>
Date:   Wed Sep 28 16:55:50 2016 -0400

    initial commit
```

10 Reference

Immutable containers:

https://github.com/Howon/Oscar/blob/master/src/native/immut/list.hpp

Ping pong Actor example:

http://alvinalexander.com/scala/scala-akka-actors-ping-pong-simple-example

General Actor Framework Inspiration

http://akka.io/docs/