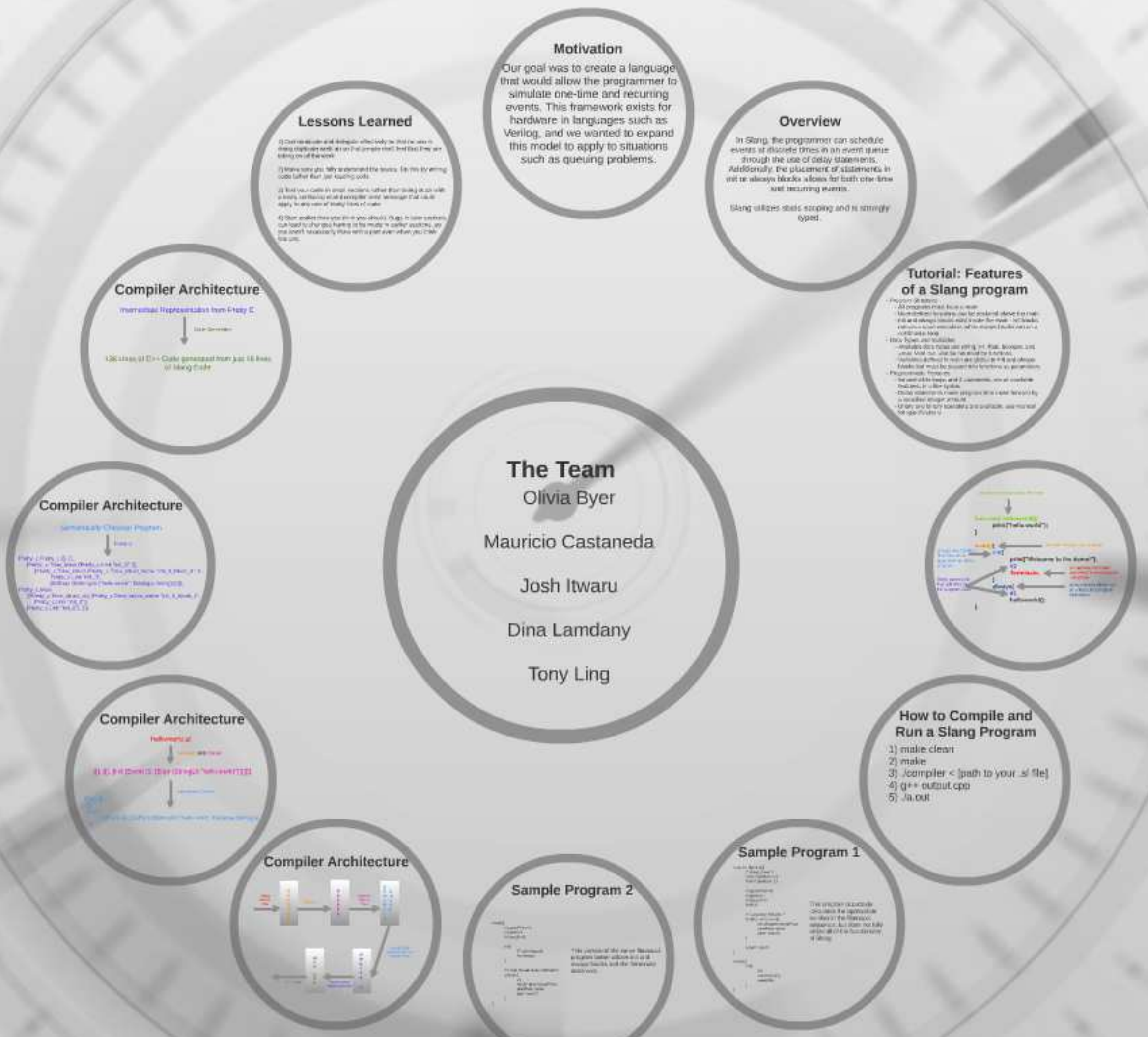
Slang: A discrete event simulation language



Motivation

Our goal was to create a language that would allow the programmer to simulate one-time and recurring events. This framework exists for hardware in languages such as Verilog, and we wanted to expand this model to apply to situations such as queuing problems.

Overview

In Slang, the programmer can schedule events at discrete times in an event queue through the use of delay statements. Additionally, the placement of statements in init or always blocks allows for both one-time and recurring events.

Slang utilizes static scoping and is strongly typed.

Tutorial: Features of a Slang program

[illegible]

Lessons Learned

[illegible]

The Team

Olivia Byer

Mauricio Castaneda

Josh Itwaru

Dina Lamdany

Tony Ling

How to Compile and Run a Slang Program

1) make clean
2) make
3) ./compiler < [path to your .sl file]
4) g++ output.cpp
5) ./a.out

Sample Program 1

[illegible]

Sample Program 2

[illegible]

Compiler Architecture

Compiler Architecture

Compiler Architecture

Compiler Architecture

Motivation

Our goal was to create a language that would allow the programmer to simulate one-time and recurring events. This framework exists for hardware in languages such as Verilog, and we wanted to expand this model to apply to situations such as queuing problems.

Overview

In Slang, the programmer can schedule events at discrete times in an event queue through the use of delay statements. Additionally, the placement of statements in init or always blocks allows for both one-time and recurring events.

Slang utilizes static scoping and is strongly typed.

Tutorial: Features of a Slang program

[illegible]

How to Compile and Run a Slang Program

1) make clean
2) make
3) ./compiler < [path to your .sl file]
4) g++ output.cpp
5) ./a.out

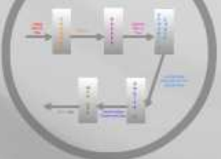
Sample Program 1

[illegible]

Sample Program 2

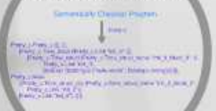
[illegible]

Compiler Architecture



Compiler Architecture

Compiler Architecture



Compiler Architecture

Lessons Learned

[illegible]

The Team

Olivia Byer

Mauricio Castaneda

Josh Itwaru

Dina Lamdany

Tony Ling

Slang: A discrete event simulation language

Motivation

The Team

Olivia Byer

Mauricio Castaneda

Josh Itwaru

Dina Lamdany

Tony Ling

Motivation

Our goal was to create a language that would allow the programmer to simulate one-time and recurring events. This framework exists for hardware in languages such as Verilog, and we wanted to expand this model to apply to situations such as queuing problems.

g

n

e

A
in

S

to
g
or
s
nd
s

Overview

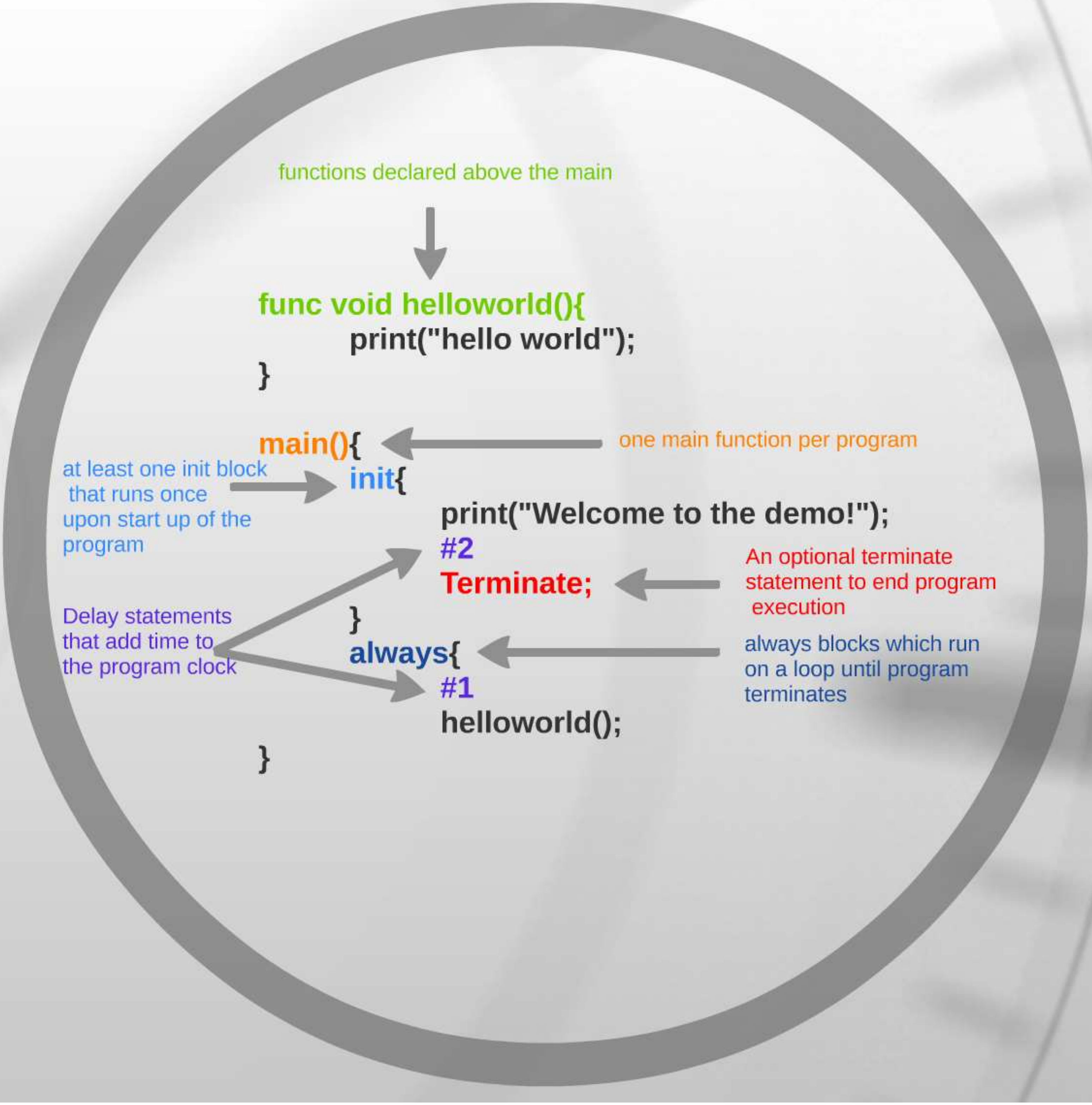
In Slang, the programmer can schedule events at discrete times in an event queue through the use of delay statements. Additionally, the placement of statements in init or always blocks allows for both one-time and recurring events.

Slang utilizes static scoping and is strongly typed.

Tutorial
of a Slan

Tutorial: Features of a Slang program

- Program Structure
  - All programs must have a main
  - User-defined functions can be declared above the main
  - init and always blocks exist inside the main - init blocks run once upon execution, while always blocks run on a continuous loop
- Data Types and Variables
  - Available data types are string, int, float, boolean, and array. Void can also be returned by functions.
  - Variables defined in main are global to init and always blocks but must be passed into functions as parameters
- Programmatic Features
  - for and while loops and if statements are all available features, in c-like syntax
  - Delay statements make program time move forward by a specified integer amount
  - Unary and binary operators are available, see manual for specifications



functions declared above the main

```
func void helloworld(){
        print("hello world");
}

main(){
    init{
        print("Welcome to the demo!");
        #2
        Terminate;
    }
    always{
        #1
        helloworld();
}
```

one main function per program

at least one init block that runs once upon start up of the program

An optional terminate statement to end program execution

Delay statements that add time to the program clock

always blocks which run on a loop until program terminates

How to Compile and Run a Slang Program

```
1) make clean
2) make
3) ./compiler < [path to your .sl file]
4) g++ output.cpp
5) ./a.out
```

gram 1

Sample Program 1

```
func int fib(int n){
        /* Base Case */
        if(n==0){return 0;}
        if(n==1){return 1;}

        int prevPrev=0;
        int prev=1;
        int result=0;
        int i=2;

        /* Calculate Results */
        for(i=2; i<=n; i++){
                result=prev+prevPrev;
                prevPrev=prev;
                prev=result;
        }

        return result;
}

main(){
        init{
                #1
                int fib=fib(7);
                print(fib);
        }
}
```

This program accurately calculates the appropriate number in the fibonacci sequence, but does not fully utilize all of the functionality of Slang

cci

e

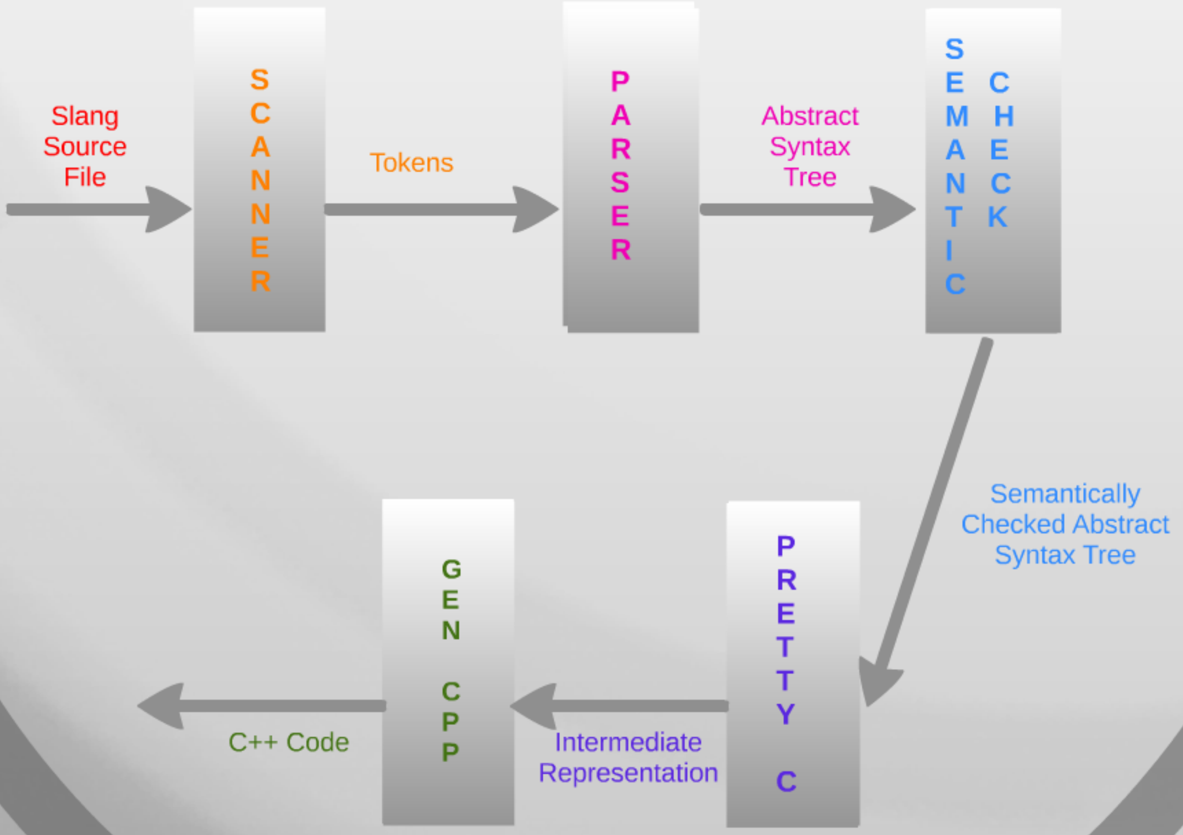
Sample Program 2

```
main(){
        int prevPrev=0;
        int prev=1;
        int result=0;

        init{
                #7 print(result);
                Terminate;
        }

        /* Loop to calculate numbers*/
        always{
                #1
                result=prev+prevPrev;
                prevPrev=prev;
                prev=result;
        }
}
```

This version of the same fibonacci program better utilizes init and always blocks and the Terminate statement.

o world", Datatype String))])]

Compiler Architecture



Slang Source File

SCANNER

Tokens

PARSER

Abstract Syntax Tree

SEMANTIC CHECK

Semantically Checked Abstract Syntax Tree

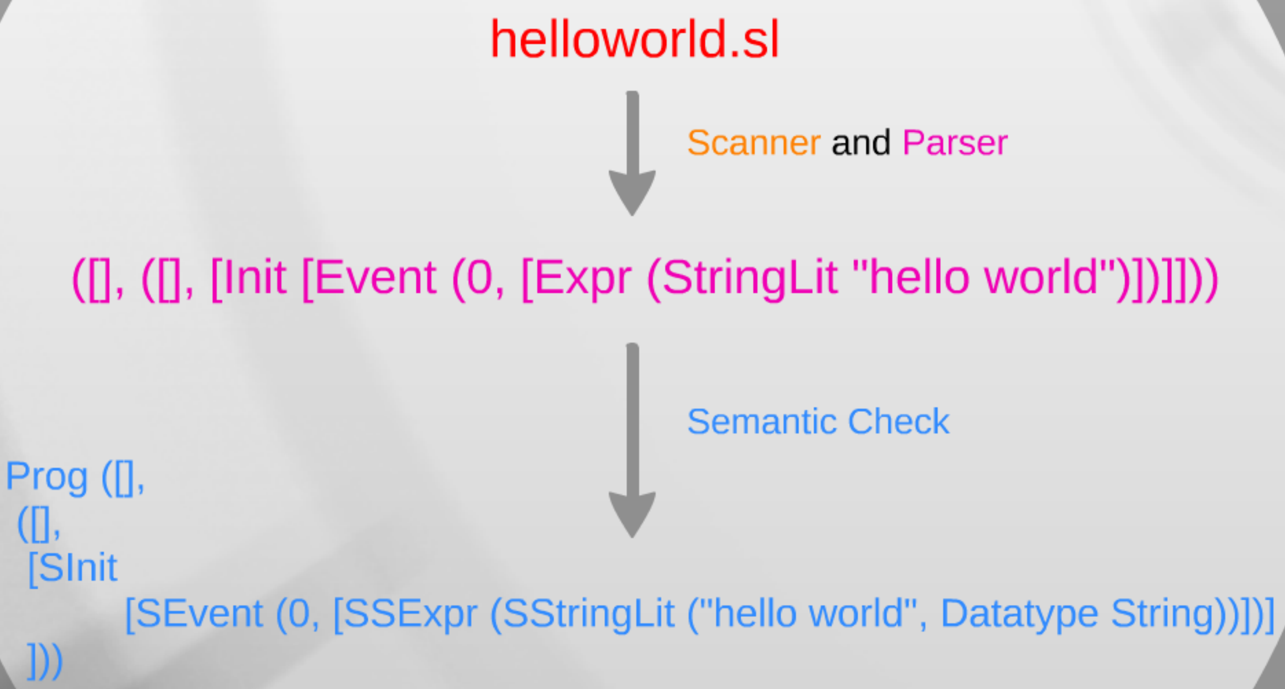
PRETTY C

Intermediate Representation

GEN CPP

C++ Code

Compiler Architecture



helloworld.sl

Scanner and Parser

```
([], ([], [Init [Event (0, [Expr (StringLit "hello world")])]]))
```

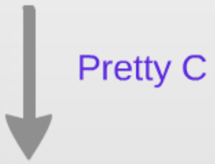
Semantic Check

```
Prog ([],
 ([],
  [SInit
          [SEvent (0, [SSExpr (SStringLit ("hello world", Datatype String))])]
  ]))
```

Compile

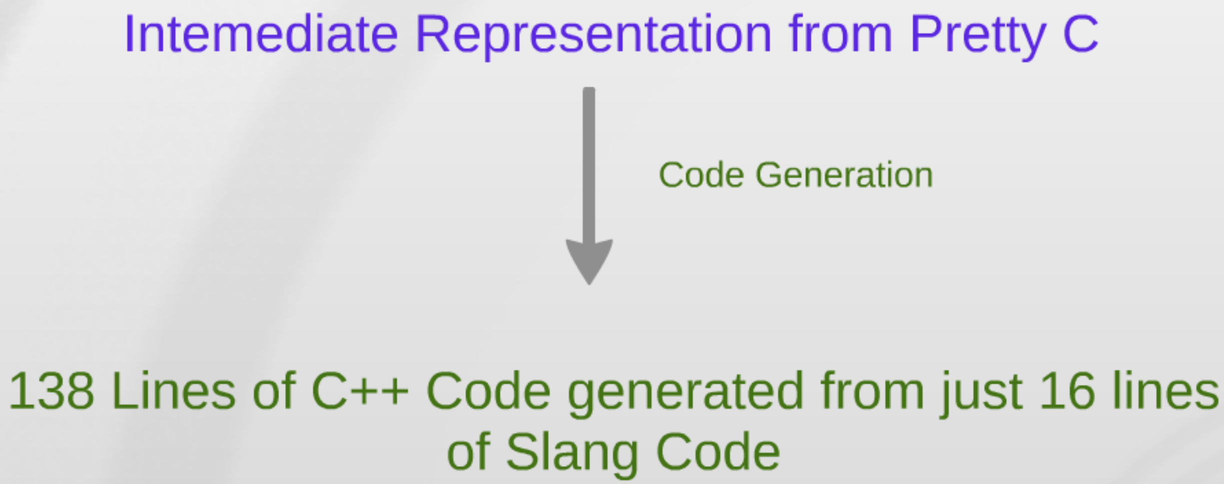
Compiler Architecture

Semantically Checked Program


Pretty C

```
Pretty_c.Pretty_c ([], [],
    [Pretty_c.Time_block (Pretty_c.Link "init_0", [],
        [Pretty_c.Time_struct (Pretty_c.Time_struct_name "init_0_block_0", 0,
                Pretty_c.Link "init_0",
                [SSExpr (SStringLit ("hello world", Datatype String))])])],
Pretty_c.Main
    ([Pretty_c.Time_struct_obj (Pretty_c.Time_struct_name "init_0_block_0",
        Pretty_c.Link "init_0")],
    [Pretty_c.Link "init_0"], []))
```

Compiler Architecture



Intemediate Representation from Pretty C

Code Generation

138 Lines of C++ Code generated from just 16 lines of Slang Code

Lessons Learned

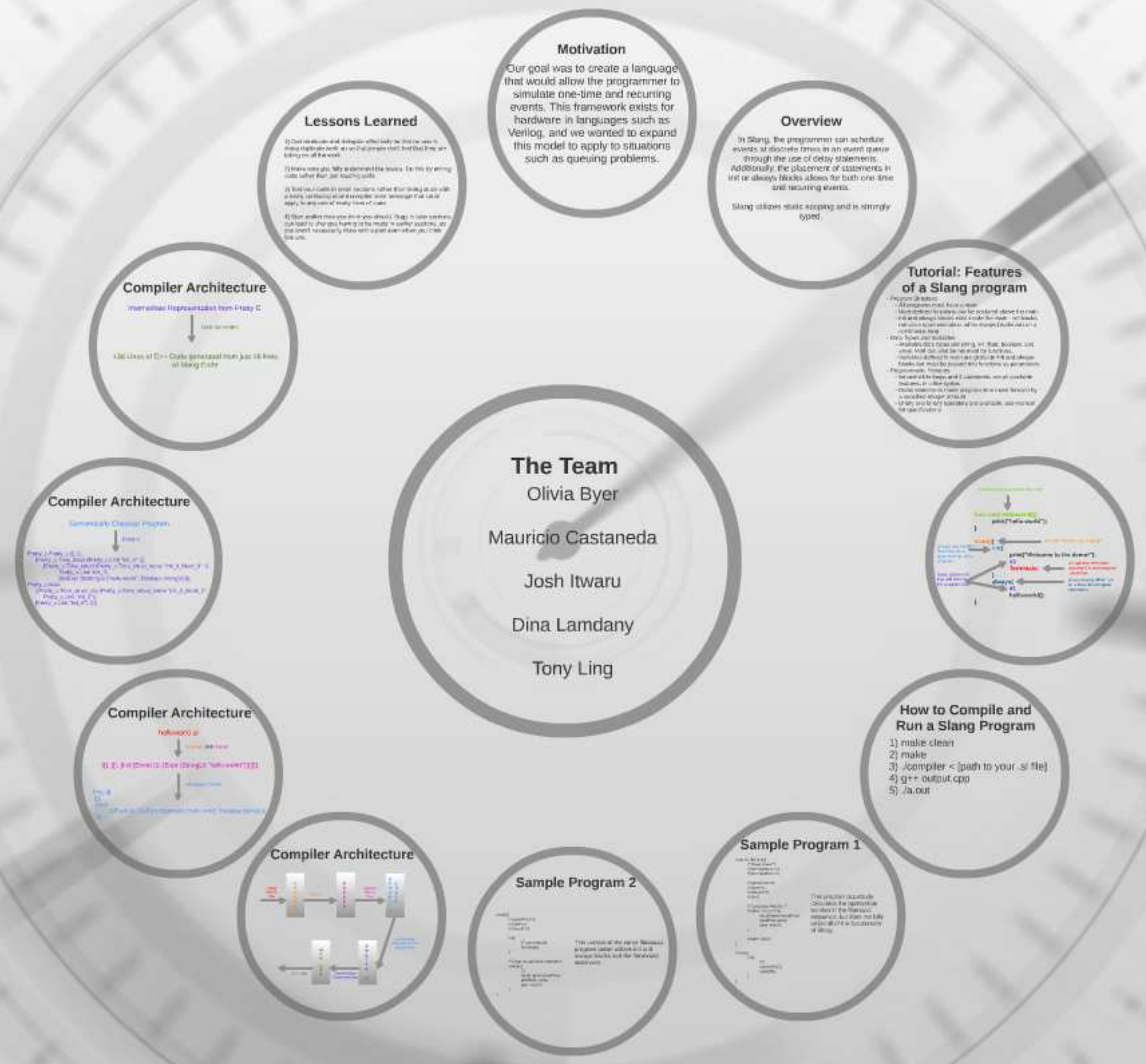
1) Communicate and delegate effectively so that no one is doing duplicate work an so that people don't feel that they are taking on all the work

2) Make sure you fully understand the basics. Do this by writing code rather than just reading code

3) Test your code in small sections rather than being stuck with a really confusing ocaml compiler error message that could apply to any one of many lines of code

4) Start earlier than you think you should. Bugs in later sections can lead to changes having to be made in earlier sections, so you aren't necessarily done with a part even when you think you are.

itecture



The Team

Olivia Byer

Mauricio Castaneda

Josh Itwaru

Dina Lamdany

Tony Ling

Motivation

Our goal was to create a language that would allow the programmer to simulate one-time and recurring events. This framework exists for hardware in languages such as Verilog, and we wanted to expand this model to apply to situations such as queuing problems.

Overview

In Slang, the programmer can schedule events at discrete times in an event queue through the use of delay statements. Additionally, the placement of statements in init or always blocks allows for both one-time and recurring events.

Slang utilizes static scoping and is strongly typed.

Tutorial: Features of a Slang program

[illegible]

```
print("hello world");
print("Welcome to the drama");
terminate;
helloworld();
```

[illegible]

How to Compile and Run a Slang Program

1) make clean
2) make
3) ./compiler < [path to your .sl file]
4) g++ output.cpp
5) ./a.out

Sample Program 1

[illegible]

Sample Program 2

[illegible]

Compiler Architecture

[illegible]

Compiler Architecture

helloworld.sl

[illegible]

Compiler Architecture

[illegible]

Compiler Architecture

[illegible]

Compiler Architecture

[illegible]

Lessons Learned

[illegible]