

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 is not as always. Models allow for both one-time and recurring events.

Slang utilizes static scoping and is strongly typed.

Tutorial: Features of a Slang Program

Program Structure

- 1) Imports and includes
- 2) Main function to drive simulation
- 3) Event queue
- 4) Delay statements
- 5) Event definitions
- 6) Event scheduling
- 7) Event execution
- 8) Event termination



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

```

#include <string>
#include <vector>
#include <map>
#include <queue>
#include <set>
#include <algorithm>
#include <math>
#include <random>
#include <time>
using namespace std;

// ... (rest of the code)

```

Sample Program 2

```

// ... (rest of the code)

```

The Team

- Olivia Byer
- Mauricio Castaneda
- Josh Itwaru
- Dina Lamdany
- Tony Ling

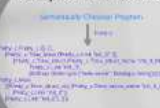
Lessons Learned

- 1) Don't make an API that is too complex. Don't try to do everything.
- 2) Make sure you're not over-engineering. Don't try to do everything.
- 3) Don't make an API that is too complex. Don't try to do everything.
- 4) Don't make an API that is too complex. Don't try to do everything.

Compiler Architecture



Compiler Architecture



Compiler Architecture



Compiler Architecture



Slang: A discrete event simulation language

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
of a Slang**

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(){ ← one main function per program
```

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

```
init{
```

```
    print("Welcome to the demo!");
```

#2

```
    Terminate;
```

← An optional terminate
statement to end program
execution

Delay statements
that add time to
the program clock

```
}
```

```
always{
```

#1

```
    helloworld();
```

← 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

Program 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

Sample Program 2

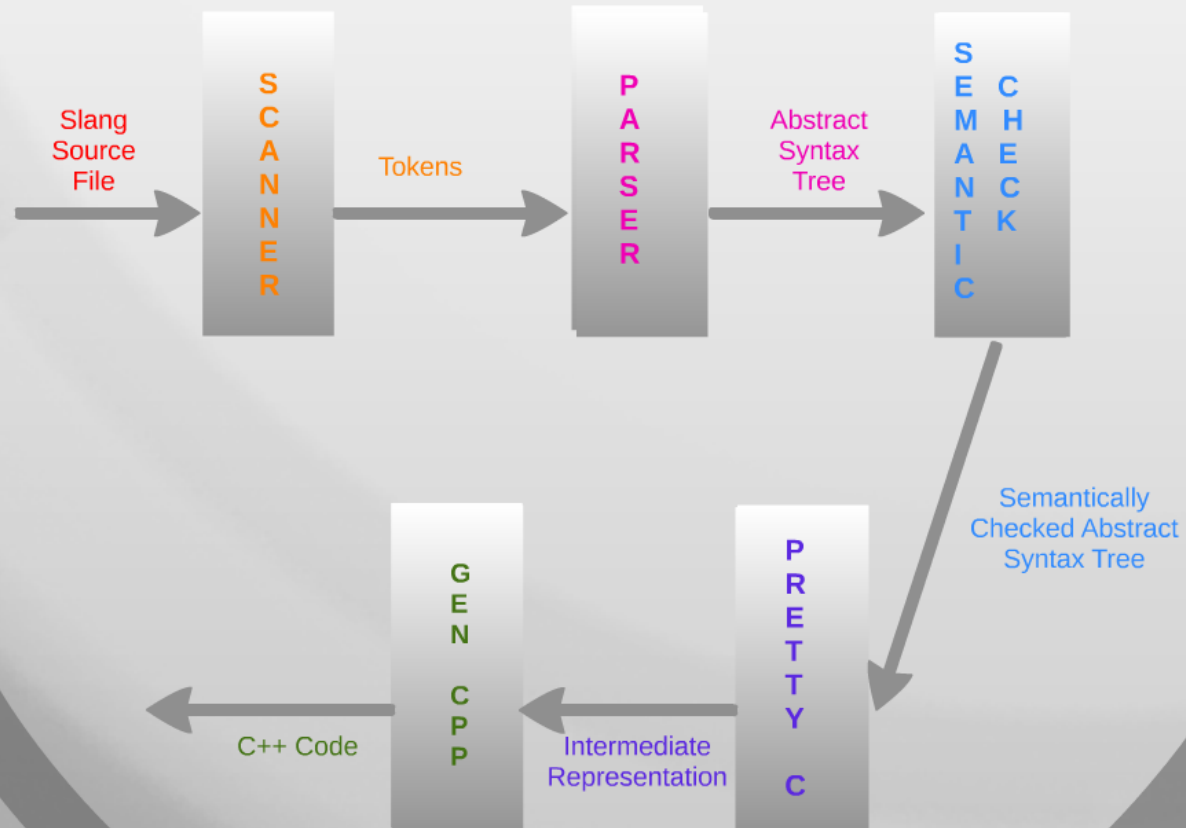
```
main(){
  int prevPrev=0;
  int prev=1;
  int result=0;

  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.

Compiler Architecture



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

Intermediate 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 and 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.

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 `wait` or `always` blocks allows for both one-time and recurring events.
 Using unique static naming and is strongly typed.

Lessons Learned

- 1) Our motivation and design objectives for this language were not clearly defined at the beginning and had to be refined as we went along.
- 2) There were a few design decisions that were not clearly defined at the beginning and had to be refined as we went along.
- 3) There were a few design decisions that were not clearly defined at the beginning and had to be refined as we went along.
- 4) There were a few design decisions that were not clearly defined at the beginning and had to be refined as we went along.

Tutorial: Features of a Slang program

Program Structure

- 1) All programs must have a main function.
- 2) The main function can be defined above or below the `main` function.
- 3) The `main` function must be defined before any other function.
- 4) The `main` function must be defined before any other function.
- 5) The `main` function must be defined before any other function.

Compiler Architecture

Intermediate Representation from C-style C

136 Lines of C++ Code generated from just 18 lines of Slang Code

Compiler Architecture

Summary of Compiler Program

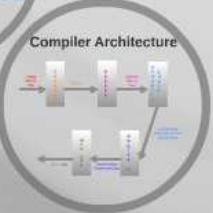
```

1) Parse the input C++ code into an Abstract Syntax Tree (AST)
2) Perform semantic analysis on the AST
3) Generate intermediate code from the AST
4) Perform optimization on the intermediate code
5) Generate final code from the intermediate code
  
```

Compiler Architecture

hlsCompiler

1) Parse the input C++ code into an Abstract Syntax Tree (AST)



Sample Program 2

```

// Sample Program 2
// This program demonstrates the use of the 'wait' statement.
// It shows how to schedule events at discrete times.
// The program will output the results of the simulation.
  
```

Sample Program 1

```

// Sample Program 1
// This program demonstrates the use of the 'always' statement.
// It shows how to schedule recurring events.
// The program will output the results of the simulation.
  
```



How to Compile and Run a Slang Program

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

The Team
 Olivia Byer
 Mauricio Castaneda
 Josh Itwaru
 Dina Lamdany
 Tony Ling