Scolang

LISTENER -> ACTION

A contract fulfillment language

Contents

1. Motivation and Background
2. Implementation Details
3. How it works
4. Development Strategy and learnings
5. Demo

Motivation and Background

- Scolang is a “Smart Contracts” based language, this means that a listener gets fulfilled and it triggers an action

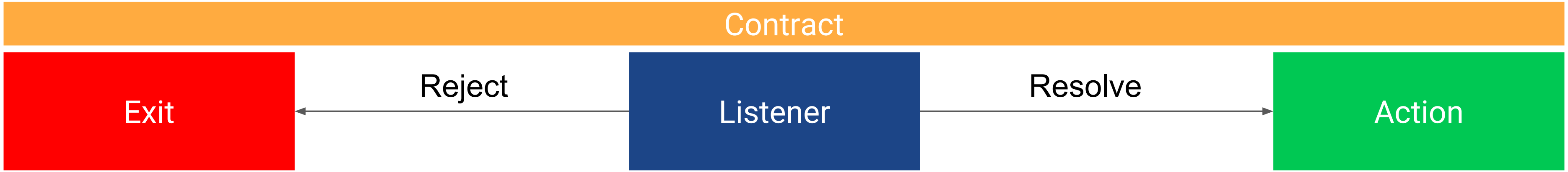
- Automate all the repetitive “IFTTT” tasks that users might have
  - Versatile use cases - IoT, Networking, Load balancing…

- Tried writing an “Alexa Skill” to turn on a phillips hue light and play some music = world of pain.

Implementation Details

1. Programming Paradigm
2. Data Types
3. Key inbuilt functions

Programming Paradigm



Declare multiple contracts in one script and they'll all be executed concurrently!

```
Listener b = { println (“I’m a listener!”); resolve; };

Action a = { println(“I’m an action!”); };

a -> b; /* This is a contract */
```

Data Types

Standard Data Types

Integer	32 bit signed integer
Boolean	1-bit Boolean variable
Float	64-bit Float
String	8 bit pointer

Scolang Types

Listener	Function Pointer
Action	Function Pointer
Contract	Integer

Key Inbuilt Functions

Purpose - Promoting IoT use case and versatility by allowing powerful contracts by having very open ended functions.

1. **Webhook(port_number)** : Opens a webhook at that port that's waiting for an input
2. **Query(query_details)** : Send a query to an endpoint of your choosing
3. **system_call(systemcall)** : Execute cmd commands on your system

Specifications

1. Statically scoped
2. Declarations must precede use/initialisation
3. Static types

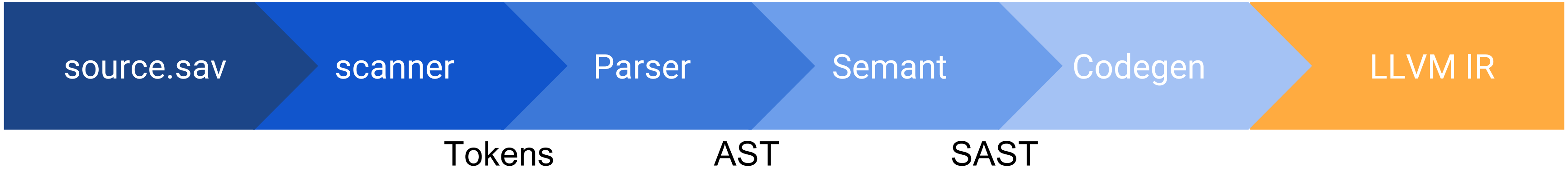
How it works : Under the hood

1. Listeners are essentially functions waiting to either die or return
2. Actions are also essentially functions
3. Whenever a contract is encountered, the listener-action pair is bound and forked into its own program, parent program returns to create more children.

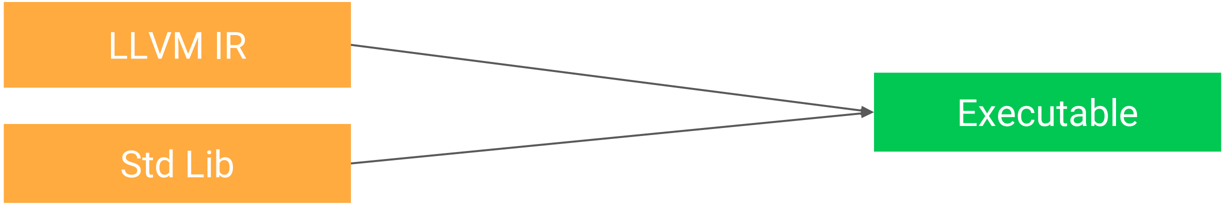
During codegen, the compiler walks through the AST, casts actions/listeners to functions and then prepares the binding by calling a C-function we wrote that manipulates the pointers to execute sequentially

Compiler Architecture

Ocaml Compiler



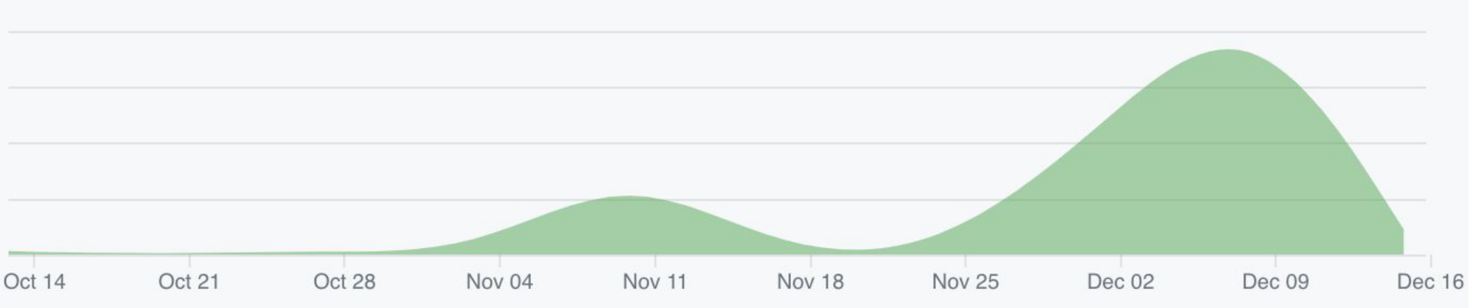
IR to Executable



Development Strategy

1. Testing in Travis CI + Shared VM for inspection
2. Environment Preservation using Docker
3. Written in OCaml, Python, C
4. Communication and Task Management via Trello and Messenger

Timeline



B-Weekly sprints before due dates

Learnings

1. Start early and use the regression testing suite as much as possible
2. Don’t waste time on things that are not the compiler(wasted a lot of time on travis CI)
3. Be less ambitious (We originally wanted to have algebraic expressions across listeners and chaining)
4. Team work makes the dream work - contribution % was near 20% for all 5 members.

Demo

- Solving the problem that initiated this entire project
  - Writing an alexa skill to turn on the lights and play some music.

Sushanth Raman	All rounder
Jackson Chen	Language Guru
Sambhav Anand	Architecture
Varun Varahabhotla	Project Manager
Kanishk Vashisht	Testing incharge