LinearAlgebra Oriented Language (LOL)

Jerry Lin, David Wan,
Shuqi Chen, Zichuan Wang

Agenda

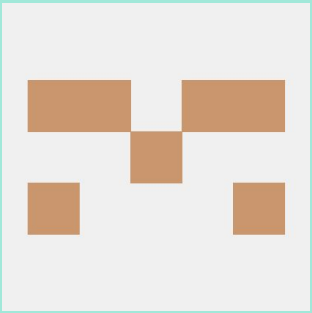
- Introduction
- Implementation
- Highlights
- Details
- Testing
- Demo
- Lessons Learned

INTRODUCTION

The LOL Team

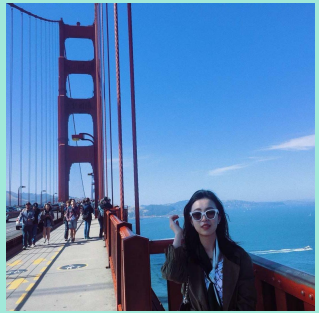
Jerry Lin

Manager



David Wan

System Architect



Shuqi Chen

Tester

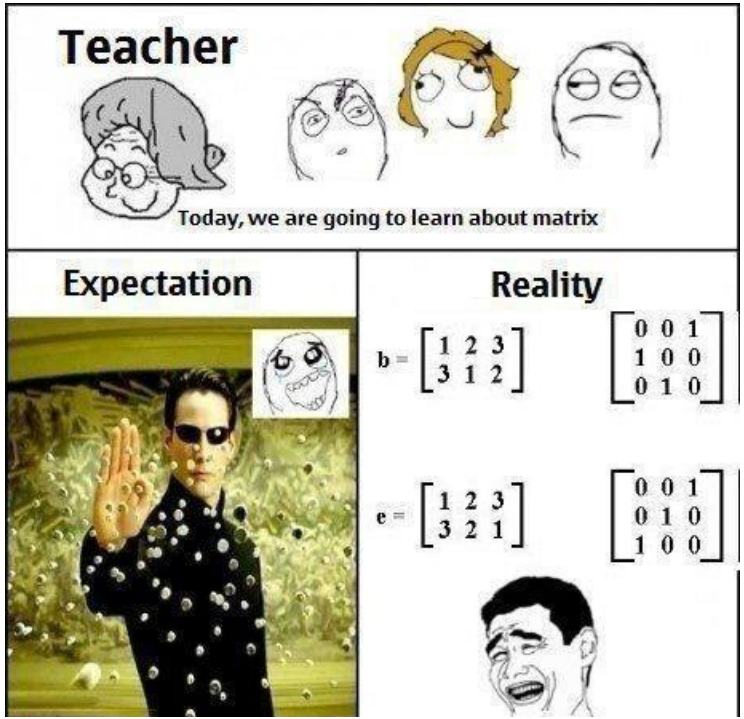
Zichuan Wang

Language Guru

Our Inspiration

- Machine Learning, Computer Vision, Quantum Computing, Robotics, etc, all heavily depend on Linear Algebra

- Functional programming can be helpful in mathematical operations

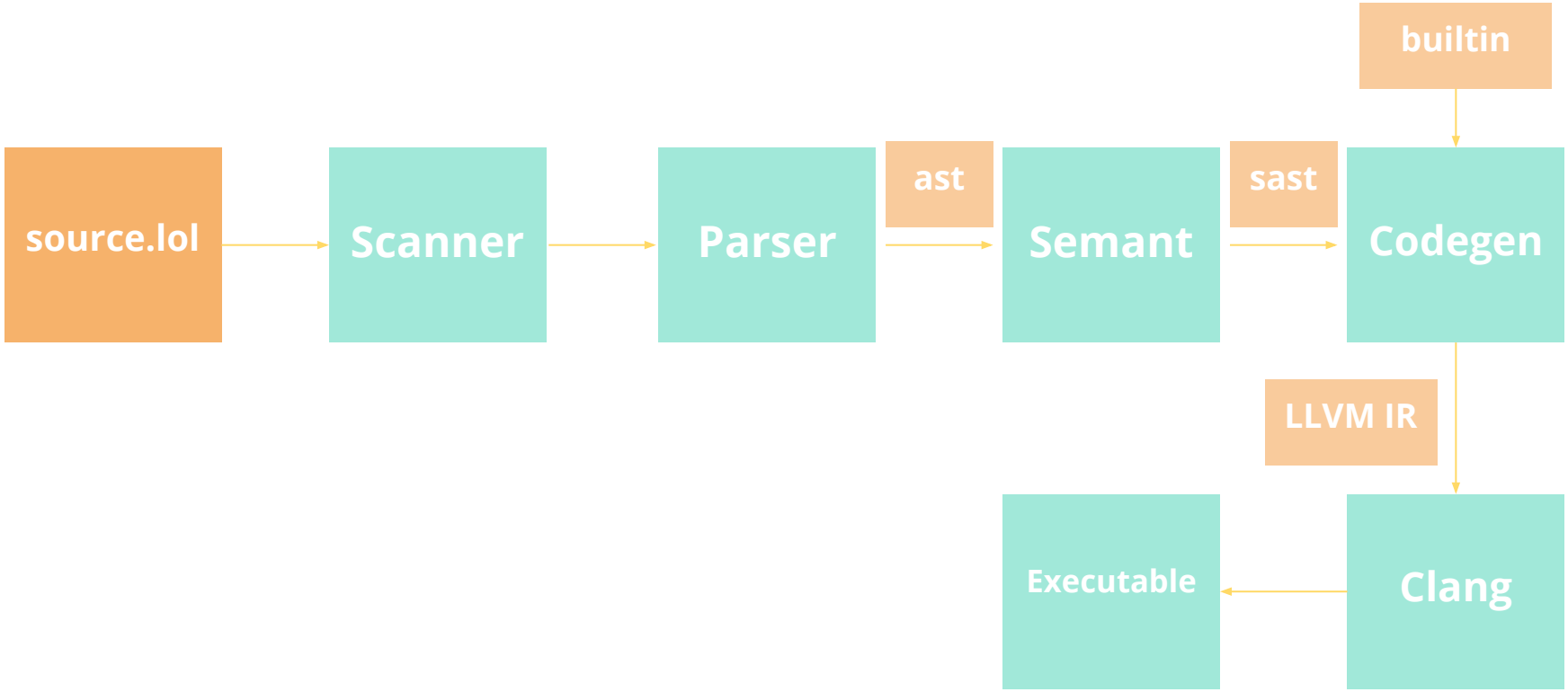


What is LOL

LOL is ...

- ~~League of Legends~~
- a linear-algebra-oriented Language
- designed for mathematicians to perform linear algebra related tasks
- more user friendly than Python/Matlab by introducing easy-to-remember math operators, such as * and @ (instead of complicated function calls)
- created with functional programming

IMPLEMENTATION

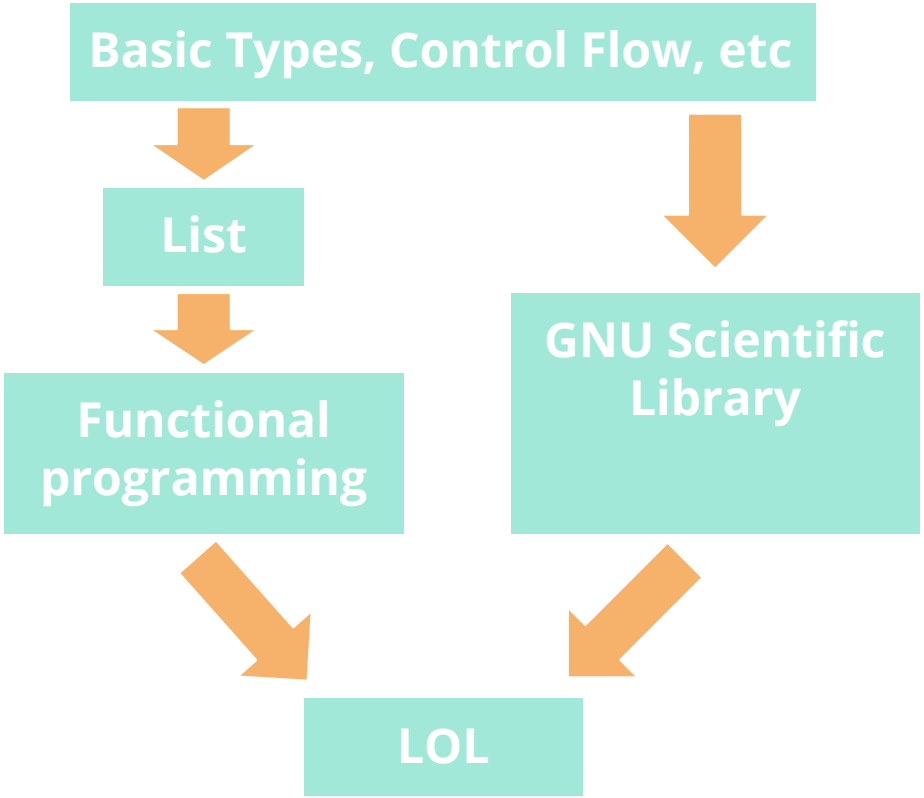

builtin
source.lol
Scanner
Parser
ast
Semant
sast
Codegen
LLVM IR
Executable
Clang

HIGHLIGHTS

- List
- Functional Programming
- Linear Algebra Support

Details

How we made LOL



Types

- LOL supports types including:
- **int:** int i = 0;
- **float:** float j = 2.2;
- **bool:** bool flag = true;
- **string:** string t = “hello world”;
- **List**
- **Matrix**

Syntax & Grammar

- LOL supports **for loops, while loops, if and else statements**, all in a C syntax
- Some valid programs include:

```
int i = 0;
while (i < 3) {
    println(str_of_int(i));
    i++;
}
```

```
bool flag = true;
if (flag){
    println("233");
}
```

```
for (int i = 0; i < 10;  ) {
    println("hello " + str_of_int(i));
    i = i + 2;
}
```

Syntax & Grammar

- LOL supports **for loops** in absence of one or more arguments
- Some valid programs include:

```
for (int i = 0; i < 10;  ) {
    println("hello " + str_of_int(i));
    i = i + 2;
}
```

```
    for ( ; ; ) {
        c = c + 1;
        if (c > 100) {
            return "done!";
        }
    }

    return "what??";
}
```

Syntax & Grammar

- LOL supports **List**
- A valid program include:

```
List<List<int>> foo = [[0,1],[2,3]];
println(str_of_int(foo[0][0]));
println(str_of_int(foo[0][1]));
println(str_of_int(foo[1][0]));
println(str_of_int(foo[1][1]));
```

```
List<int> a = [34, 235, 2534, 435];
println(str_of_int(a.length()));
```

```
List<int> a = [1, 2, 3];
a.append(4);
println(str_of_int(a[3])
```

Syntax & Grammar

- LOL supports **first-class functions**
- LOL supports List<func>
- A valid program include:

```
func func(int,int:int) apply_f(func(int:int) f) {
    return func int (int i, int j) {
        return f(i) + f(j);
    };
}

func int double(int x) {
    return x * 2;
}

func(int,int:int) sum_of_double = apply_f(double);

println(str_of_int(sum_of_double(3, 5)));
```

```
func int addOne (int num){
    return num + 1;
}

func int addTwo (int num){
    return num + 2;
}
List<func(int:int)> lst;

lst.append(addOne);
lst.append(addTwo);

println(str_of_int(lst[0](1)));
println(str_of_int(lst[1](1)));
```

Syntax & Grammar

- LOL supports **Matrix struct and corresponding computations**
- Some valid programs include:

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
Matrix t2 = Matrix([2.0, 3.0, 4.0], [4.0, 5.0, 6.0]);
t1.dive(t2);
t1.addc(2.0);
mswapr(t1, 0, 1);
printm(t1);
```

```
Matrix t3 = mgetc(t1, 1);
Matrix t4 = mtrans(t1);
Matrix t5 = mgetsub(t1, 0, 0, 0, 0);
t3 = mtrans(t3);
printm(t3);
```

Matrix Operators & Functions

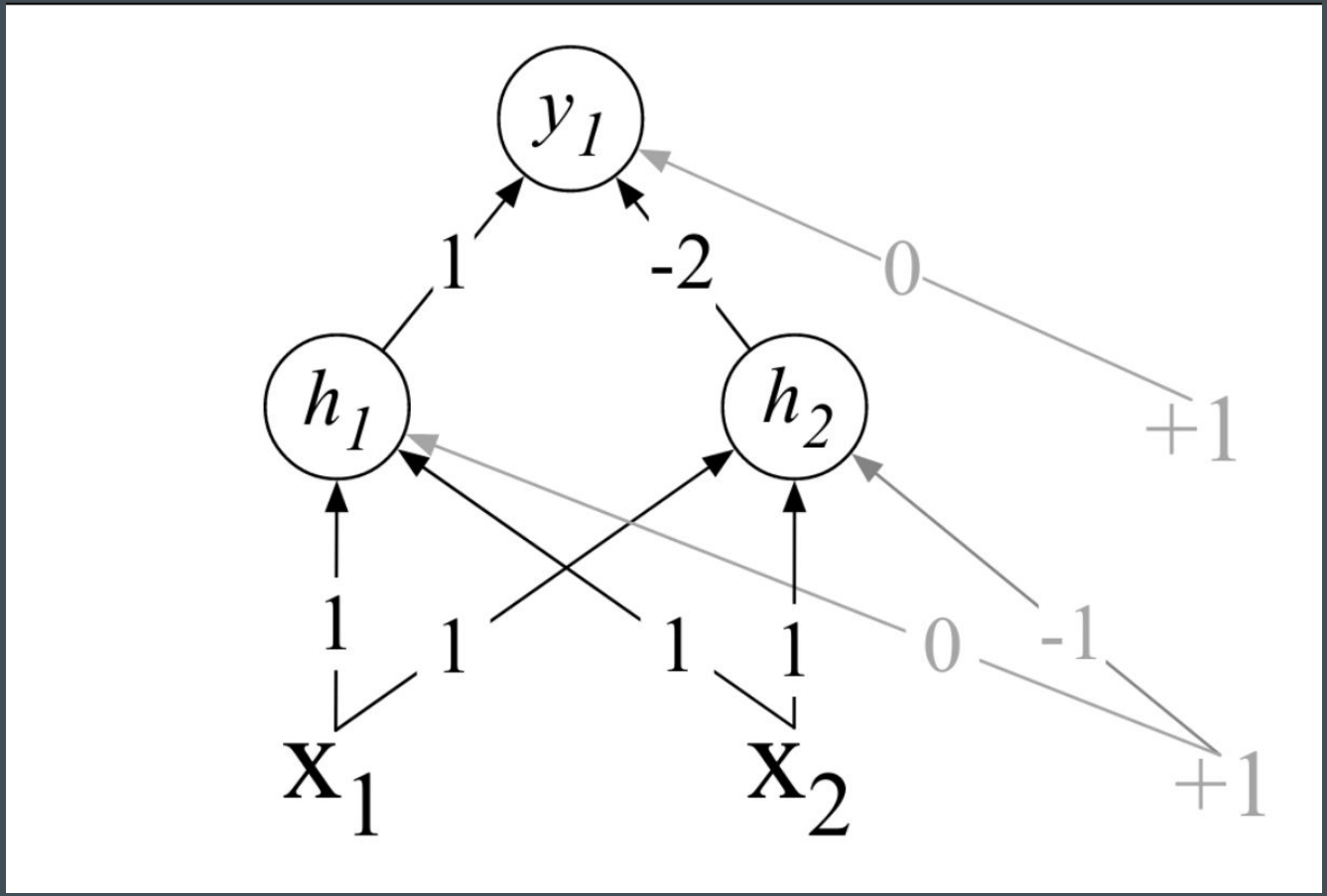
- **Declaration**: Matrix([1.0, 2.0], [1.0, 2.0]), minit()
- **Getter & Setter**: .get(), .set(), mget(), mset()
- **Addition & Subtraction**: .add(), .sub(), madd(), msub()
- **Multiplication & Division**: mmulc(), maddc(), mmule(), mdive(), mdot(), mmul()
- **Exponent & Logarithm**: mexpe(), mloge()
- **Swap & Copy & Transpose**: mswapr(), mswapc(), mtrans(), mcopy()
- **Matrix View**: mgetr(), mgetc(), mgetsub()

TESTING

Test Suite

- Sample program output compared to *.out file
- Check the following file types: fail-* for semant tests and llvm/runtime tests
- The output of fail-* varies from platform to platform
- Generally 1-2 tests per feature
- Over 70 tests in total

DEMO TIME



XOR problem after Goodfellow, I., Bengio, Y., and Courville, A. (2016).DeepLearning. MIT Press.

Lessons Learned

What did we learn?

- Definitely Ocaml
- Break a huge project down to small tasks
- Became a fan of Visual Studio Code

Thank you!