
sinful.cats

Objective oriented toy language compiler to LLVM on C++

Team 1-206
Innopolis University, 2019
Compilers Construction

Problem domain

Develop a complete compiler for a predefined grammar of object-oriented language.

Chosen constraints: C++ and LLVM.

Team

- Andrew Volkov
- Gleb Petrakov
- Alexander Solovyov
- Elvira Salikhova

make

```
syntax: parser/syntax.yy
    $(BISON) parser/syntax.yy --output=parser/syntax.cc

compile:
    $(CC) $(CFLAGS) $(LLVM_CFLAGS) -fexceptions -c -g \
        parser/syntax.cc parser/syntax.hh parser/location.hh \
        parser/driver.cc parser/driver.hh \
        lexer/lexer.cc lexer/lexer.hh \
        lexer/checkers.cc lexer/checkers.hh \
        parser/ast.cc parser/ast.hh \
        parser/generator.cc parser/generator.hh \
        main.cc
```

Infrastructure

- Makefiles to build and test the system.
- A few test files in toy language.
- Docker image.
- System documentation.

Problem domain

Develop a complete compiler for a predefined grammar of object-oriented language.

Chosen constraints: C++ and LLVM.

No any competences in any of it...

Problem domain

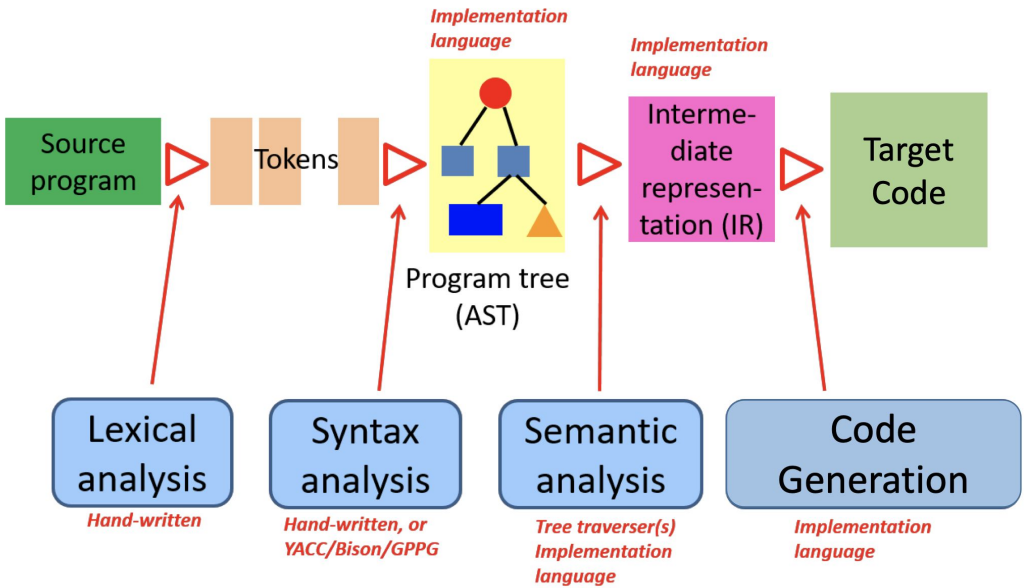
Develop a complete compiler for a predefined grammar of object-oriented language.

Chosen constraints: C++ and LLVM.

No any competences in any of it…

The hardest!

Pipeline (is a problem, actually)



Things got done

- ✓ Lexical analysis (custom)
- ✓ Syntax analysis (Bison)
- ✓ Semantic actions (Bison + custom)
- ⚠️ Intermediate representation (LLVM)

Lexical analysis

```
parser::symbol_type parse_symbol(Driver &driver) {
    char c;
    driver.file.get( &: c);

    if (c == '(') {
        return parser::make_LEFT_PARENTHESIS(std::move(driver.location));
    }
    if (c == ')') {
        return parser::make_RIGHT_PARENTHESIS(std::move(driver.location));
    }
    if (c == '[') {
```

Symbol parsing part of code

- Simple handwritten lexer to parse input per symbol.
- Defines yylex() to use it in Bison parser.

Syntax analysis

```
Node::Node(std::string value, ClassName class_name) {
    this->value = value;
    this->class_name = class_name;
    this->next = NULL;
}
```

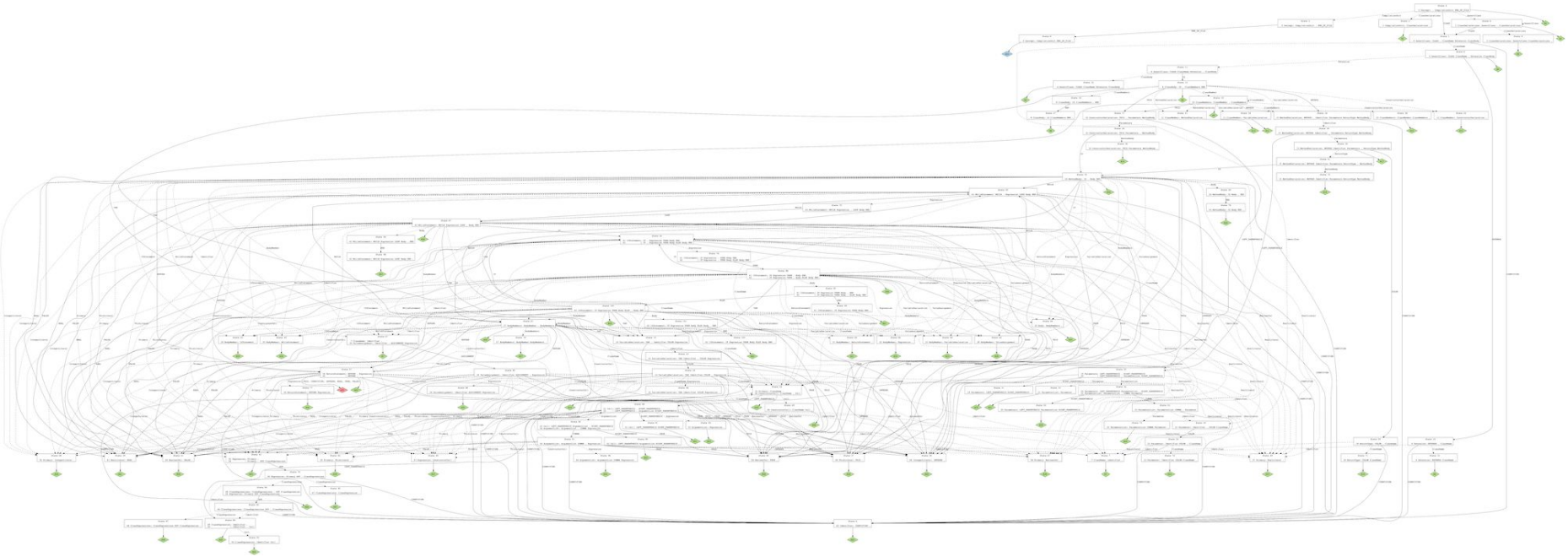
Syntax node structure

Bison-based parser with almost no interesting moments.

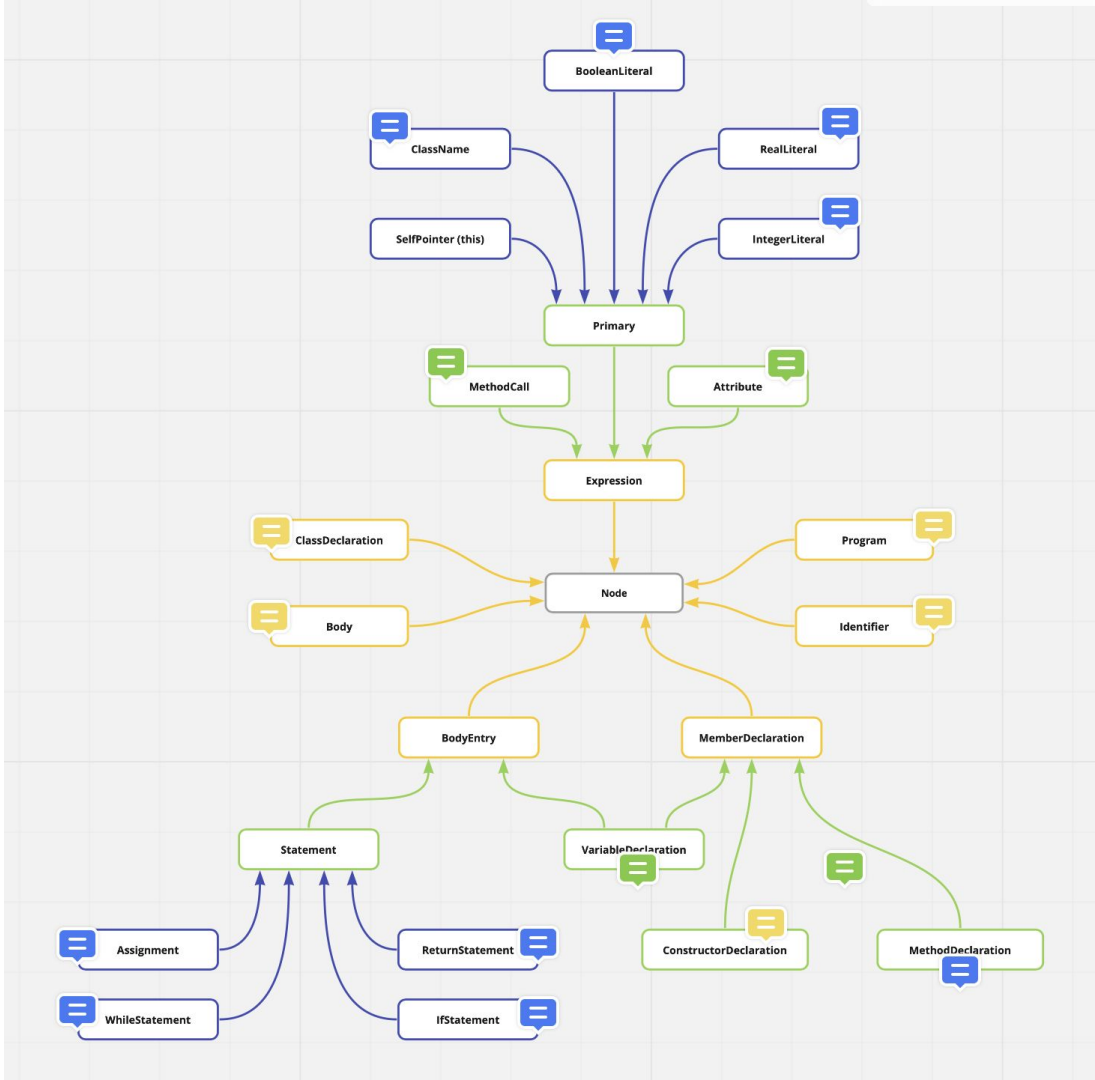
Parser output example

```
Processing class Human
Found constructor declaration
Found variable declaration: status
Found variable declaration: knowLanguage
Processing class Student
Found constructor declaration
Found variable declaration: averagePoints
Found variable declaration: averageSleep
Processing class Employee
Found constructor declaration
Found variable declaration: loveWork
Found variable declaration: salary
Processing class Chief
Found constructor declaration
Found variable declaration: goodChief
```

Tree… (Graphviz, 10428×3667)



AST


BooleanLiteral
ClassName
RealLiteral
SelfPointer (this)
IntegerLiteral
Primary
MethodCall
Attribute
Expression
ClassDeclaration
Program
Node
Body
Identifier
BodyEntry
MemberDeclaration
Statement
VariableDeclaration
Assignment
ReturnStatement
ConstructorDeclaration
MethodDeclaration
WhileStatement
IfStatement

Bison

How Bison feels like

Bison

How Bison feels like

Bison

How Bison feels like

Actually, great documentation and nice examples.

The part
where he kills you

Problems (2)

There are like 2-3 examples of compilers and all of them be like:

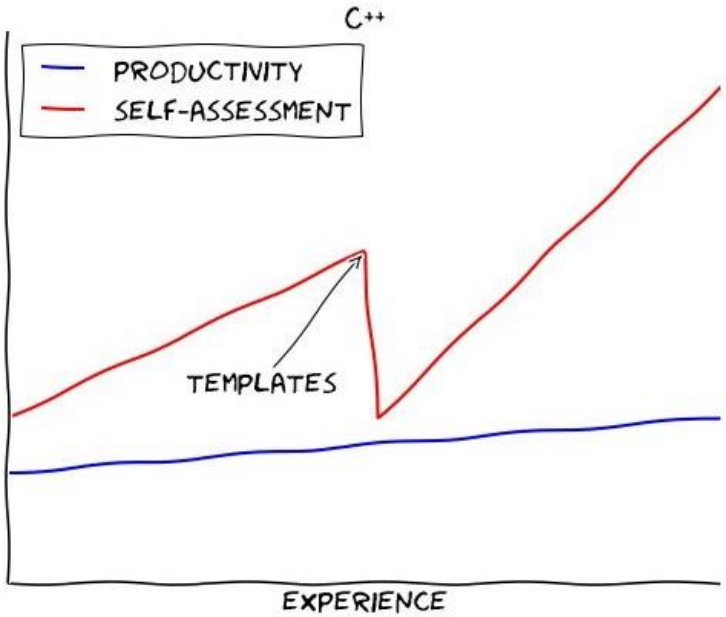
2 + 2 = 4. Obviously,

$$|\mathrm{Aut}(P)| = \prod_{k=1}^{n}(p^{d_k} - p^{k-1}) \prod_{j=1}^{n}(p^{e_j})^{n-d_j} \prod_{i=1}^{n}(p^{e_i-1})^{n-c_i+1}.$$

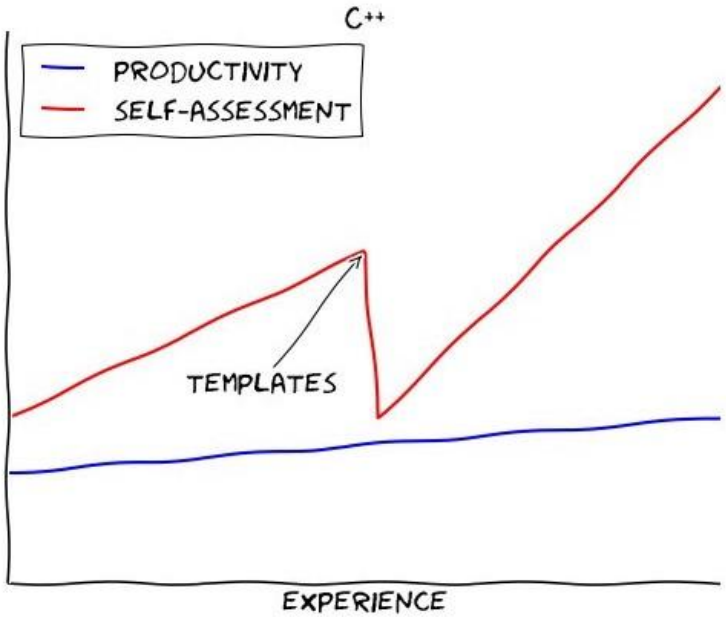
Problems (3)

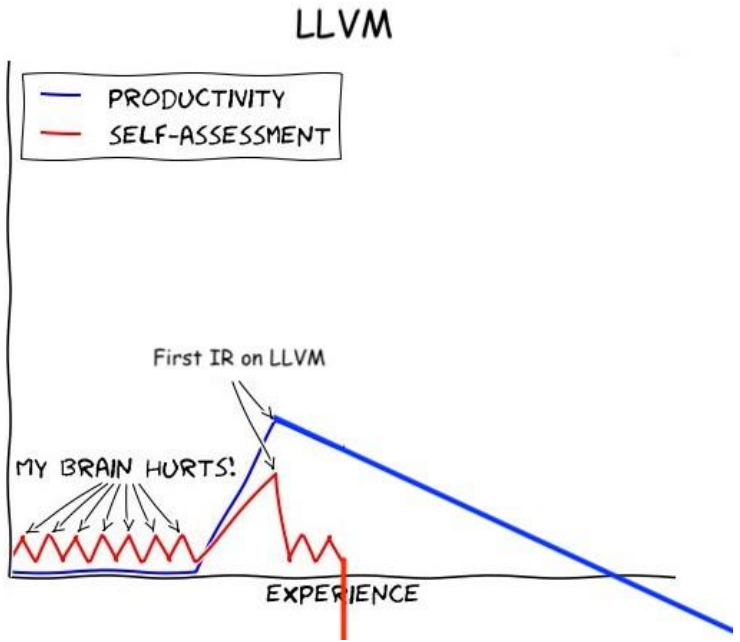
How LLVM feels like

LLVM learning curve



LLVM learning curve

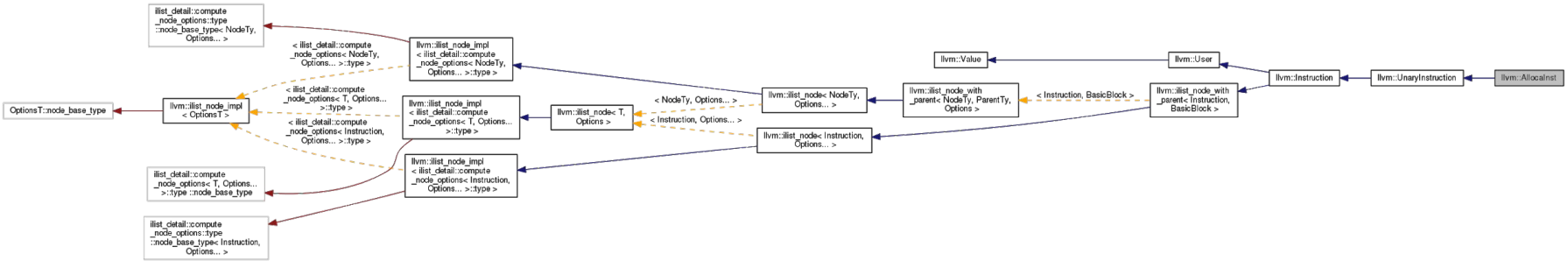




Here we go again

There are, like, 3-4 active LLVM versions
without backwards compatibility,
none of them have comprehensive documentation,
none of them have comprehensive “real world apps”.

LLVM class diagram (official)



Tutorial LLVM builder example

```
1240    // Create a subprogram DIE for this function.
1241    DIFile *Unit = DBuilder->createFile(KSDbgInfo.TheCU->getFilename(),
1242                                        KSDbgInfo.TheCU->getDirectory());
1243    DIScope *FContext = Unit;
1244    unsigned LineNo = P.getLine();
1245    unsigned ScopeLine = LineNo;
1246    DISubprogram *SP = DBuilder->createFunction(
1247        FContext, P.getName(), StringRef(), Unit, LineNo,
1248        CreateFunctionType(TheFunction->arg_size(), Unit), ScopeLine,
1249        DINode::FlagPrototyped, DISubprogram::SPFlagDefinition);
1250    TheFunction->setSubprogram(SP);
```

Our LLVM bytecode example

```
%Animal = type { %Boolean*, %Real*, %Integer*, %Boolean*, %Integer* }
%Real = type { %Real* }
%Boolean = type { %Boolean* }
%Integer = type { %Integer* }
%Chief = type { %Employee*, %Boolean* }
%Employee = type { %Human*, %Boolean*, %Integer* }
%Human = type { %Animal*, %Integer*, %Boolean* }
%Student = type { %Human*, %Real*, %Integer* }

define void @main() {
entry:
  %Animal = alloca %Animal
  %Boolean = alloca %Boolean
  %Chief = alloca %Chief
  %Employee = alloca %Employee
  %Human = alloca %Human
  %Integer = alloca %Integer
  %Real = alloca %Real
  %Student = alloca %Student
  ret void
}
```

Nice references

Writing Your Own Toy Compiler Using Flex, Bison and LLVM
https://gnuu.org/2009/09/18/writing-your-own-toy-compiler/

My First Language Frontend with LLVM Tutorial
https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/


It ain't much, but it's honest work

EXAMPLES