中山大學 SUN YAT-SEN UNIVERSITY

计算机学院 COMPUTER SCIENCE AND ENGINEERING

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS292：编译器构造实验(Compiler Construction)

Task2：语法分析 ANTLR

Yat Compiler Construction with AI

yatcc-ai.com

YatCC团队

中山大学 计算机学院

国家超级计算广州中心

2026.3.22

www.nscc-gz.cn

实验介绍

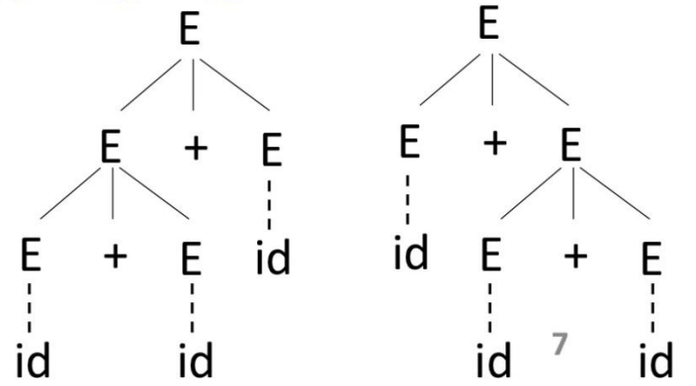
语法分析器

提取出语法结构!

build > test > task2 > functional-0 > 000_main.sysu.c > {} answer.simplied.json > ...

```json
{
  "kind": "TranslationUnitDecl",
  "inner": [
    {
      "kind": "FunctionDecl",
      "name": "main",
      "type": {
        "qualType": "int ()"
      },
      "inner": [
        {
          "kind": "CompoundStmt",
          "inner": [
            {
              "kind": "ReturnStmt",
              "inner": [
                {
                  "kind": "IntegerLiteral",
                  "type": {
                    "qualType": "int"
                  },
                  "valueCategory": "prvalue",
                  "value": "3"
                }
              ]
            }
          ]
        }
```

id + id + id



answer.txt

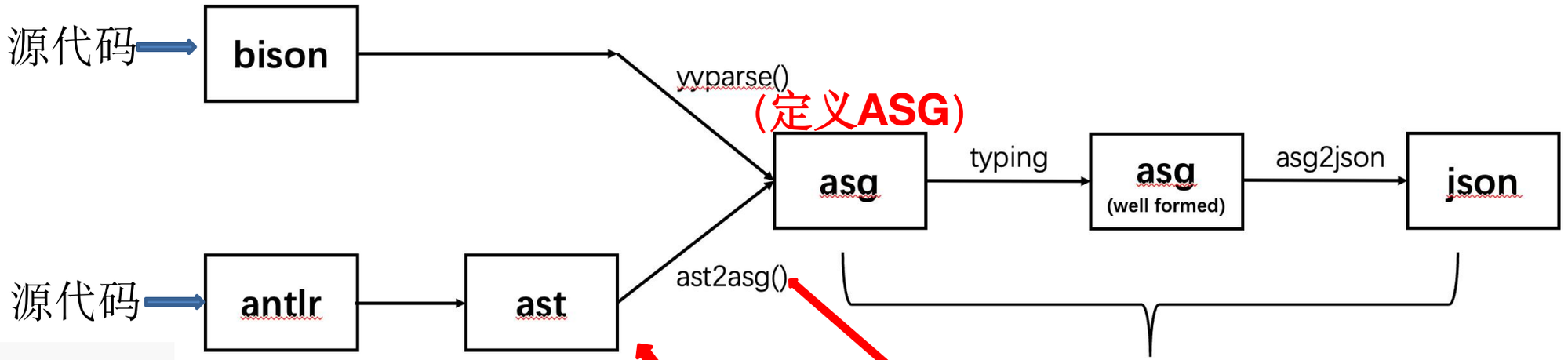
build > test > task2 > functional-0 > 000_main.sysu.c > answer.txt

```
TranslationUnitDecl 0x564c458c67d8 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x564c458c7008 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
| `-BuiltinType 0x564c458c6da0 '__int128'
|-TypedefDecl 0x564c458c7078 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
| `-BuiltinType 0x564c458c6dc0 'unsigned __int128'
|-TypedefDecl 0x564c458c7380 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_ta
| `-RecordType 0x564c458c7150 'struct __NSConstantString_tag'
|   `-Record 0x564c458c70d0 '__NSConstantString_tag'
|-TypedefDecl 0x564c458c7428 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
| `-PointerType 0x564c458c73e0 'char *'
|   `-BuiltinType 0x564c458c6880 'char'
|-TypedefDecl 0x564c458c7720 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'struct __va_list_tag[1]'
| `-ConstantArrayType 0x564c458c76c0 'struct __va_list_tag[1]' 1
|   `-RecordType 0x564c458c7500 'struct __va_list_tag'
|     `-Record 0x564c458c7480 '__va_list_tag'
`-FunctionDecl 0x564c45918d28 </YatCC/test/cases/functional-0/000_main.sysu.c:1:1, line:3:1> line:1:5 main 'int ()'
  `-CompoundStmt 0x564c45918e48 <col:11, line:3:1>
    `-ReturnStmt 0x564c45918e38 <line:2:5, col:12>
      `-IntegerLiteral 0x564c45918e18 <col:12> 'int' 3
```

实验内容-ANTLR

实验一：源代码 → ANTLR → Token流

实验二：




```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
   |-- Ast2Asg.cpp
   |-- Ast2Asg.hpp
   `-- SYsUParser.g4
-- common
   `-- asg.hpp
```

(将AST转换为ASG)

(定义AST)

实验内容-ANTLR

词法分析

1.不启动复活：

Token流（实验一Output） → 实验二语法分析器 → AST

2.启动复活（<u>推荐且默认</u>）：

源代码（实验零Output） → 实验二词法分析器 → Token流 → 实验二语法分析器 → AST

需要填充

```
antlr4::ANTLRInputStream input(inFile);
SYsULexer lexer(&input);

antlr4::CommonTokenStream tokens(&lexer);
SYsUParser parser(&tokens);

auto ast = parser.compilationUnit();
```

实验内容-ANTLR

填充实验二的词法分析部分

```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

SYsULexer.tokens.hpp：
由SYsULexer.py
根据SYsULexer.tokens生成
在SYsULexer.hpp中引用

task > 2 > antlr > SYsULexer.tokens

```
Int=1
Identifier=2
LeftParen=3
RightParen=4
Return=5
RightBrace=6
LeftBrace=7
Constant=8
Semi=9
Equal=10
Plus=11
Minus=12
Comma=13
LeftBracket=14
RightBracket=15
```

build > task > 2 > antlr > SYsULexer.tokens.hpp

```
#include <cstdint>

namespace SYsULexerTokens {

constexpr size_t kInt = 1;
constexpr size_t kIdentifier = 2;
constexpr size_t kLeftParen = 3;
constexpr size_t kRightParen = 4;
constexpr size_t kReturn = 5;
constexpr size_t kRightBrace = 6;
constexpr size_t kLeftBrace = 7;
constexpr size_t kConstant = 8;
constexpr size_t kSemi = 9;
constexpr size_t kEqual = 11;
constexpr size_t kPlus = 12;
constexpr size_t kComma = 13;
constexpr size_t kLeftBracket = 14;
constexpr size_t kRightBracket = 15;
```

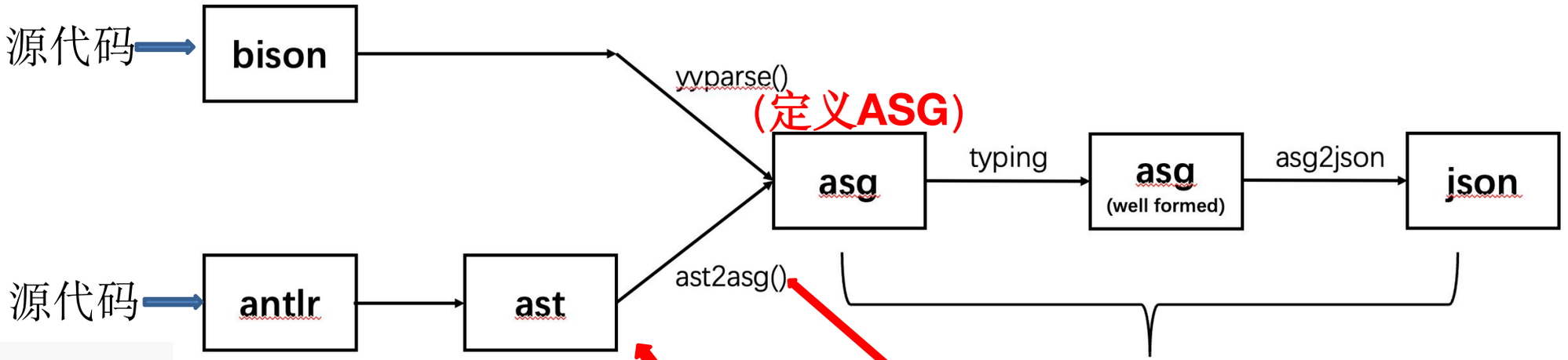
task > 2 > antlr > SYsULexer.cpp > ...

```
static const std::unordered_map<std::string, size_t> kClangTokens{
  { "eof", antlr4::Token::EOF },
  { "int", kInt },
  { "identifier", kIdentifier },
  { "l_paren", kLeftParen },
  { "r_paren", kRightParen },
  { "return", kReturn },
  { "r_brace", kRightBrace },
  { "l_brace", kLeftBrace },
  { "numeric_constant", kConstant },
  { "semi", kSemi },
  { "equal", kEqual },
  { "plus", kPlus },
  { "minus", kMinus },
  { "comma", kComma },
  { "l_square", kLeftBracket },
  { "r_square", kRightBracket }
};
```

填写映射表：
关键字是antlr中的词法token
值是在实验二中使用的词法token
（可以借机重命名为方便的名字）

实验内容-ANTLR

实验二：




```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
   |-- Ast2Asg.cpp
   |-- Ast2Asg.hpp
   `-- SYsUParser.g4
-- common
   `-- asg.hpp
```

(将AST转换为ASG)

(定义AST)

实验内容-ANTLR

定义AST

只要通过修改SYsUParser.g4定义好AST，
ANTLR就会自动完成语法解析的工作!

```
-- antlr
   |-- Ast2Asg.cpp
   |-- Ast2Asg.hpp
   `-- SYsUParser.g4
-- common
   `-- asg.hpp
```

task > 2 > antlr > SYsUParser.g4

```
27  unaryExpression
28      :   postfixExpression
29      |   unaryOperator unaryExpression
30      ;
31
32  unaryOperator
33      :   Plus | Minus
34      ;
35
36  multiplicativeExpression
37      :   unaryExpression ((Star|Slash|Mod) unaryExpression)*
38      ;
39
40  additiveExpression
41      :   multiplicativeExpression ((Plus|Minus) multiplicativeExpression)*
42      ;
```

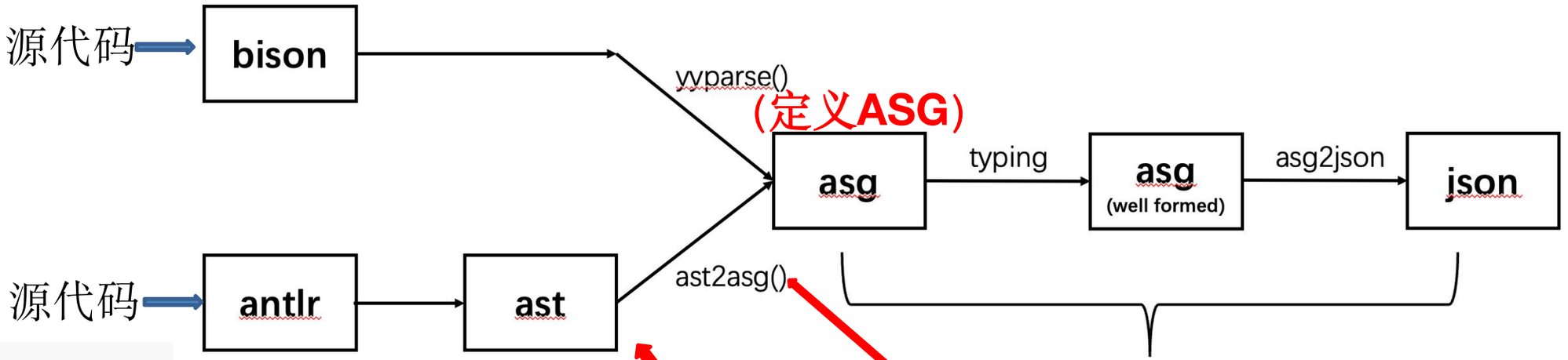
```
antlr4::ANTLRInputStream input(inFile);
SYsULexer lexer(&input);

antlr4::CommonTokenStream tokens(&lexer);
SYsUParser parser(&tokens);

auto ast = parser.compilationUnit();
```

实验内容-ANTLR

实验二：




```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
   |-- Ast2Asg.cpp
   |-- Ast2Asg.hpp
   `-- SYsUParser.g4
-- common
   `-- asg.hpp
```

(将AST转换为ASG)

(定义AST)

实验内容-ANTLR

From AST to ASG

通过设置将AST转换为ASG的任务，
考察同学们对语法解析过程的理解!

ASG由asg.hpp定义，
因此要重点理解asg.hpp。
asg.hpp将在教程的“公用代码”部分讲解

小贴士：

1. 在ast2asg.cpp中定义了新函数记得在ast2asg.hpp声明
2. 看完“实验思路”中尤其是“上手思路”一节你就可以上手做实验了。不过想要正确理解与完成整个实验，教程中的其它部分也是很重要的
3. 使用agent进行实验分析

```
-- antlr
  |-- Ast2Asg.cpp
  |-- Ast2Asg.hpp
  `-- SYsUParser.g4
-- common
  `-- asg.hpp
```

```
41  auto ast = parser.compilationUnit();
42  Obj::Mgr mgr;
43
44  asg::Ast2Asg ast2asg(mgr);
45  auto asg = ast2asg(ast->translationUnit());
```

实验内容-ANTLR

当实验卡住时，我们可以做什么

1. 调试！在ast2asg.cpp中添加断点，使用 ctx->getText() 打印内容。调试将出现两种情况：
   a) 打印发现ast中缺失所需内容：SYsUParser.g4没写好，导致AST中不存在所需的节点。
      ① 检查SYsUParser.g4中是否存在左递归、二义性等问题。
      ② 对比/build/test/task2/中的answer.txt，检查是否有遗漏的情况没有实现。
   b) 打印发现ast中存在所需内容，但输出的JSON文件中没有所需内容：ast2asg.cpp没有写好。
2. 阅读实验文档！没有头绪可能是对实验的理解不够，尤其是关于ASG的部分。除了ASG的介绍，你还可以从Typing类、Asg2Json类中找找思路。
3. 使用agent！可以借助agent的功能来分析代码中出现的问题。

谢　谢！