Decaf 课程项目中的 LL(1) 语法分析

朱俸民

清华大学《编译原理》课程助教

2019 年 8 月

目录

目录

Decaf 语言示例


```
1  class Deck {
2      int current;
3      int[] cards;
4      class rndModule rnd;
5
6      void Init(class rndModule rnd) {
7          cards = new int[52];
8          this.rnd = rnd;
9      }
10
11     void Shuffle() {
12         for (current = 1; current <= 52; current = current + 1)
13             cards[current - 1] = current % 13;
14         current = current - 1; // ...
15     } // ...
16 }
```

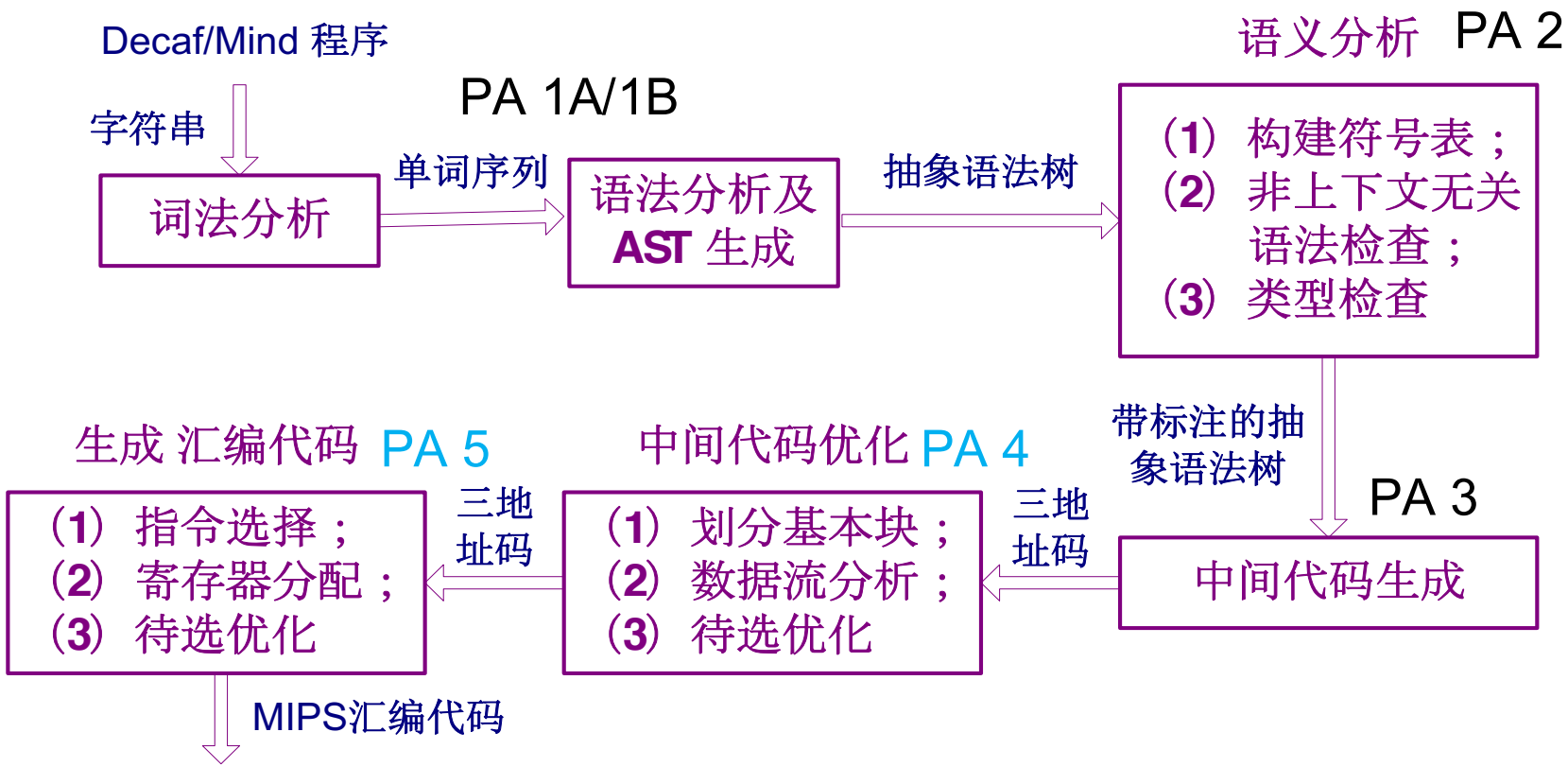
Decaf 语言特性

- 静态强类型：整数、布尔、字符串、类、数组
- 控制流程：条件、循环
- 面向对象：单继承、动态分发
- 类 Java 的命令式语言，特性简单，语义明确

Decaf 历史

- 始于计算机系 98 级本科生《编译原理》课
- 基于 Stanford University CS143
- 03 – 04 级：开发语言由 C++ 变成 Java
- 05 级姚班：单趟扫描改为多趟扫描（Mind，C++）
- 05 级至今：基于 Mind 扩展若干语言特征（Java）

Decaf 项目框架



注：蓝色标注的阶段为选做

真实世界的编译器

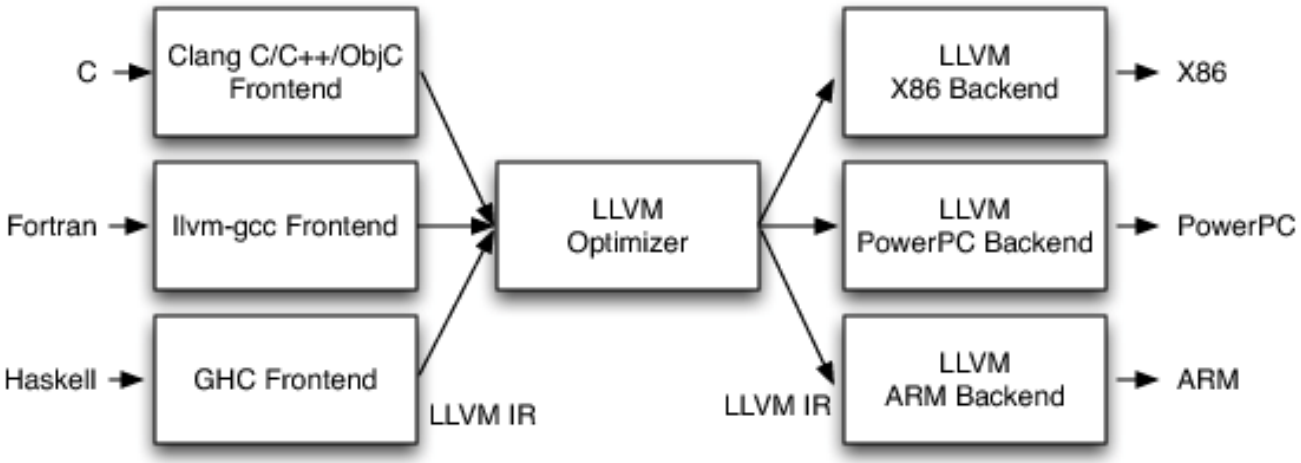


Figure: LLVM's Implementation of the Three-Phase Design, `http://www.aosabook.org/en/llvm.html`

The Scala compiler phases: `scala -Xshow-phases`.

学生任务

新特性实现，如：

- Repeat-until 语句（14’）
- 复数类型（17’）
- Super 表达式（17’）
- 本地类型推断（18’）
- 数组扩展：解析、迭代等（18’）
- Lambda 表达式（19’，计划）

扩展实验（18’）

基于 Decaf 框架或者自行开发，部分成果：

- 函数重载（支持子类型、默认值参数）
- 异常处理
- First-class functions（支持子类型）
- 解释器 + 编译器（LLVM 后端）
- Rust 版 Decaf 编译器

目录

语法解析器构造

两种主流方法：

- Parser generator
  - Lex + Yacc (LR)
  - Antlr (LL)

语法解析器构造

两种主流方法：

- Parser generator
  - Lex + Yacc (LR)
  - Antlr (LL)
- Parser combinators library
  - Parsec (Haskell)
  - FastParse (Scala)

曾经的 PA1-B

手写 LL(1) Parser

LL1-Parser-Gen

- https://github.com/paulzfm/LL1-Parser-Gen

LL1-Parser-Gen

- https://github.com/paulzfm/LL1-Parser-Gen
- 类 Yacc 的语法描述格式

LL1-Parser-Gen

- https://github.com/paulzfm/LL1-Parser-Gen
- 类 Yacc 的语法描述格式
- 可嵌入语义动作

LL1-Parser-Gen

- https://github.com/paulzfm/LL1-Parser-Gen
- 类 Yacc 的语法描述格式
- 可嵌入语义动作
- 冲突检查与报告

LL1-Parser-Gen

- https://github.com/paulzfm/LL1-Parser-Gen
- 类 Yacc 的语法描述格式
- 可嵌入语义动作
- 冲突检查与报告
- 生成 Java 源码

语法描述


```
1  // E -> E '+' E {left} | T
2  E   : T E1
3      {
4          $$.expr = $1.expr;
5          for (Term t : $2.terms) {
6              $$.expr = new ArithExpr(t.op, $$.expr, t.expr);
7          }
8      }
9  ;
10 E1  : '+' T E1
11     {
12         $$.terms.add(new Term(Expr.ADD, $2.expr));
13         $$.terms.addAll($3.terms);
14     }
15     | /* empty */
16 ;
```

代码生成


```
1  private SemValue parseE() throws Exception {
2    switch (lookahead) {
3      case NUM: case '(':
4      { SemValue[] params = new SemValue[3];
5        params[0] = new SemValue();
6        params[1] = parseT();
7        params[2] = parseE1();
8        params[0].expr = params[1].expr; // user action starts
9        for (Term t : params[2].terms)
10         params[0].expr = new ArithExpr(t.op,params[0].expr,t.expr);
11       return params[0]; }
12     default:
13     { String[] acc = {name(NUM), name('(')};
14       throw error(name(lookahead), acc); }
15   }
16 }
```

课程版本

为支持 PA1B 错误恢复，该版本不生成完整的 Java 源码，而生成以下元信息：

- First 集合、Follow 集合
- 分析预测表
- 语义动作

供在顶层框架中调用相应接口使用

优势

无需关心计算的细节，侧重设计错误恢复的算法

错误恢复框架


```
1  private SemValue parse(int symbol) {
2      Set<Integer> beginSym = beginSet(symbol);
3      Set<Integer> followSet = followSet(symbol);
4
5      if (!beginSym.contains(lookahead)) { /* error and recovery */ }
6
7      Map.Entry<Integer, List<Integer>> result = query(symbol,
           lookahead);
8      int actionId = result.getKey();
9      List<Integer> right = result.getValue();
10     boolean success = true;
11     for (int s : right) { /* parse recursively */ }
12
13     if (success) { /* semantic actions */ }
14     else return null;
15 }
```

工具演示

- 生成解析器 Java 源码
- 冲突提示
- 实验框架

目录

展望

- Decaf 框架多语言支持（Java/Rust/Scala）
- LL1-Parser-Gen 多语言代码生成支持
- Decaf 框架多前端/后端支持
  （Antlr/JVM/LLVM/CLR/RISC-V）
- 引入新语言特性（Kotlin/Scala/Rust/Haskell）
- 周边：Language Server/REPL/Static Analysis/Reasoning
- 探索编译器的开发框架和“设计模式”

课程组成员

- 教师：王生原（wwssyy@mail.tsinghua.edu.cn）、陈渝、姚海龙
- 近期助教：沈游人、甄艳洁、朱俸民、冀伟清、戴臻旸、王润基、李晨昊

谢谢大家！

Q & A

https://paulz.me/files/decaf-ll1.pdf