编译原理课程设计报告 I

语法分析器自动生成器

小组成员:
计算机 1202 张艺瀚 (组长)
计算机 1202 马鹏程
计算机 1202 马海琴

本模块负责人:
张艺瀚

指导教师:
朱靖波

January 8, 2015

语法分析器自动生成器

计算机 1202 张艺瀚
学号:20123852

January 8, 2015

摘要

摘要: 本模块实现了 3 个语法分析器的自动生成:SLR(1)(使用 LR(0) 自动机),LR(1) 和 LL(1).

首先对输入文法进行变换, 我们使用计算理论中的严格的文法变换方法处理文法:

1. 去除无用符号
2. 去除 ε 产生式
3. 去除单一产生式
4. 化为 CNF(Chomsky Normal Form)
5. 化为 GNF(Greibach Normal Form)

这部分内容与课程设计题目无关, 不做介绍. 对于符合各个语法分析器要求的文法, 首先计算 FIRST 和 FOLLOW 集合, 然后分别为其生成相应的语法分析器.

对于 SLR(1), 使用 LR(0) 自动机, 需要计算每个项目集合的闭包 closure 和其在各个文法符号下的转换 goto, 以此为依据计算出所有项目集合: 项目集合族, 这样就可以生成 SLR(1) 分析表了. 我们为 SLR(1) 分析表编写表驱动程序, 为给定文法分析字符串, 验证设计.

对于 LR(1), 使用 LR(1) 自动机, 与 SLR(1) 类似, 只需为每个项目计算向前看符号 lookaheadSymbol, 并在计算转移 goto 和生成分析表时考虑它即可. 我们也为 LR(1) 分析表编写了表驱动程序, 为给定文法分析字符串, 验证设计.

对于 LL(1), 利用文法的 FIRST 和 FOLLOW 集合即可直接生成分析表. 同样, 为 LL(1) 分析表编写表驱动程序, 为给定文法分析字符串, 验证设计.

根据测试结果比较各个语法分析方法的优劣势和性能.

关键字:SLR(1),LR(1),LL(1), 文法, 自动机, 语法分析

Contents

1 概述

本模块实现了 3 个语法分析器的自动生成:SLR(1)(使用 LR(0) 自动机),LR(1) 和 LL(1). 对于给定文法, 先计算 FIRST 和 FOLLOW 集合. 然后:

1. SLR(1): 计算每个项目集合的闭包 closure 和其在各个文法符号下的转换 goto, 以此为依据计算出所有项目集合: 项目集合族, 最终生成 SLR(1) 分析表.
2. LR(1): 与 SLR(1) 类似, 只需为每个项目计算向前看符号 lookaheadSymbol, 并在计算转移 goto 和生成分析表时考虑它即可.
3. LL(1): 利用文法的 FIRST 和 FOLLOW 集合直接生成分析表.

为每种分析器的分析表编写对应的表驱动程序, 为给定文法分析字符串, 验证设计并比较性能.

2 课程设计任务及要求

2.1 设计任务

分析表的自动生成算法的设计实现.

2.2 设计要求

1. 在深入理解编译原理基本原理的基础上, 对于选定的题目, 以小组为单位, 先确定设计方案;
2. 设计系统的数据结构和程序结构, 设计每个模块的处理流程. 要求设计合理;
3. 编程序实现系统, 要求实现可视化的运行界面, 界面应清楚地反映出系统的运行结果;
4. 确定测试方案, 选择测试用例, 对系统进行测试;
5. 运行系统并要通过验收, 讲解运行结果, 说明系统的特色和创新之处, 并回答指导教师的提问;
6. 提交课程设计报告.

3 算法及数据结构

3.1 算法的总体思想

对于任意给定的文法, 先计算其 FIRST 和 FOLLOW 集合, 然后针对不同的分析方法生成分析表并使用配套的表驱动程序进行测试.

SLR(1) 和 LR(1) 均要先计算闭包 closure, 转移 goto 和项目集合族 (LR(1) 在计算过程中还要考虑向前看符号 lookaheadSymbol) 才能生成分析表;LL(1) 直接利用文法信息即可直接生成分析表.

3.2 计算文法的 FIRST 和 FOLLOW 集合

3.2.1 功能

计算任意给定文法的 FIRST 和 FOLLOW 集合.

3.2.2 数据结构

FIRST 和 FOLLOW 集合数据结构见附录 8(1).

3.2.3 算法

见附录 9. 计算 FIRST 集合算法 (1); 计算 FOLLOW 集合算法 (2).

3.3 SLR(1)

3.3.1 功能

计算项目集合的闭包 closure 和在文法符号下的转移 goto, 以此求得项目集合族, 建立分析表. 也实现了配套的表驱动程序.

3.3.2 数据结构

SLR(1) 分析器数据结构见附录 8(2).

3.3.3 算法

见附录 9. 计算闭包 closure 算法 (3); 计算转移 goto 算法 (4); 计算项目集合族算法 (5); 分析表生成算法 (6); 表驱动算法 (7).

3.4 LR(1)

3.4.1 功能

计算项目集合的闭包 closure(项目中考虑向前看符号 looaheadSymbol) 和在文法符号下的转移 goto, 以此求得项目集合族, 建立分析表. 也实现了配套的表驱动程序.

3.4.2 数据结构

LR(1) 分析器数据结构见附录 8(3).

3.4.3 算法

见附录 9. 计算闭包 closure 算法 (8); 计算转移 goto 算法 (9); 计算项目集合族算法 (10); 分析表生成算法 (11); 表驱动算法 (12).

3.5 LL(1)

3.5.1 功能

利用文法的 FIRST 和 FOLLOW 集合直接建立分析表并实现了配套的表驱动程序.

3.5.2 数据结构

LL(1) 分析器数据结构见附录 8(4).

3.5.3 算法

见附录 9. 分析表生成算法 (13); 表驱动算法 (14).

4 程序设计与实现

4.1 程序说明

1. 程序使用 C++ 编写, 大量使用 C++11 和 C++14 新特性, 开发环境为 Ubuntu 12.04 LTS, 编译器版本为 gcc 4.9.2.
2. 程序输入从格式化文件中读入, 输出打印在终端内.
3. 用 e 表示空串 ε, 不同于其他文法符号, 程序中做特殊处理.
4. 计算 FIRST 和 FOLLOW 集合时, 为了避开迭代算法实现时对求值顺序的考虑, 将算法改造为迭代至不动点的形式.
5. SLR(1) 和 LR(1) 使用增广文法, 即添加产生式 $S' \to S$, 注意此时 FIRST 和 FOLLOW 集合需重新计算.
6. 将 \$ 作为初始时分析栈栈顶符号.

4.2 实验结果

语法分析器对下面的 2 个测试语法正确运行.

文法 1:

$$
\begin{aligned}
&E \to TA \\
&A \to +TA|-TA|\varepsilon \\
&T \to FB \\
&B \to *FB|/FB|\varepsilon \\
&F \to (E)|0|1|2|3|4|5|6|7|8|9
\end{aligned}
\tag{1}
$$

文法 2:

$$
\begin{aligned}
&E \to E+T|E-T|T \\
&T \to T*F|T/F|F \\
&F \to (E)|0|1|2|3|4|5|6|7|8|9
\end{aligned}
\tag{2}
$$

对于不含左递归的文法 1 和含左递归的文法 2, 均正确计算出 FIRST 和 FOLLOW 集合,SLR(1),LR(1) 语法分析器均可正确生成分析表, 对于不含左递归的文法 1,LL(1) 语法分析器可正确生成分析表. 利用自动生成的分析表,SLR(1),LR(1) 和 LL(1) 语法分析器均可正确解析下面的书写正确的复杂的一位数含括号四则运算表达式:

```
1+(1-3*(2-(8+9-7*6/2)/3+9+(1+3)-2)+1)/4
```

均不能正确解析书写错误的表达式 (最后一个 4 之前缺少一个运算符):

```
1+(1-3*(2-(8+9-7*6/2)/3+9+(1+3)-2)+1)4
```

5 结论

通过实现该设计并进行测试, 可以清楚的看到,LR(1) 分析表由于考虑了向前看符号而比 SLR(1) 分析表大. 二者作为自底向上的分析方法, 可以容忍带有左递归的文法, 而 LL(1) 作为自顶向下的分析方法, 则不允许文法中出现左递归.

6 参考文献

美 Alfred V.Aho Ravi Sethi Jeffrey D. Ullman 著. 李建中, 姜守旭译.《编译原理》. 北京: 机械工业出版社.2003.

7 收获、体会和建议

本设计完全依照 *Compilers Principles, Techniques and Tools second Edition* 实现, 与课堂内容可能存在出入. 事实证明课堂讲授的很多理论出处不明且不具有可操作性.

8 附录: 数据结构

```
vector<set<char>> first; // for symbols in VN and VT, elements in
    VT are followed by elements in VN
vector<set<char>> follow; // only for symbols in VN
```

Listing 1: FIRST 和 FOLLOW 集合数据结构

```
class LR0Item
{
public:
  LR0Item() = default;
  LR0Item(unsigned int myLProductionRuleIdx, unsigned int
   myRProductionRuleIdx, unsigned int myDotPos);

  unsigned int lProductionRuleIdx;
  unsigned int rProductionRuleIdx;
  unsigned int dotPos;

  friend bool operator < (const LR0Item& i1, const LR0Item& i2);
  friend bool operator ==(const LR0Item& i1, const LR0Item& i2);
  friend bool operator > (const LR0Item& i1, const LR0Item& i2);
  friend bool operator !=(const LR0Item& i1, const LR0Item& i2);

private:

};

class LR0ActionTabItem
{
public:
  LR0ActionTabItem() = default;
  LR0ActionTabItem(const Action& myAction, unsigned int
   myLProductionRuleIdx, unsigned int myRProductionRuleIdx,
   unsigned int myItemSetIdx);

  Action action;
  unsigned int lProductionRuleIdx;
  unsigned int rProductionRuleIdx;
  unsigned int itemSetIdx;
```

```
};

vector<set<LR0Item>> lr0ItemSetFamily;
vector<vector<LR0ActionTabItem>> lr0ActionTab;
vector<vector<unsigned int>> lr0GoToTab;
vector<unsigned int> lr0Stack;
string lr0Input;
```

Listing 2: SLR(1) 分析器数据结构

```
class LR1Item
{
public:
  LR1Item() = default;
  LR1Item(const LR0Item& myLR0Item, char myLookaheadSymbol);
  LR1Item(unsigned int myLProductionRuleIdx, unsigned int
   myRProductionRuleIdx, unsigned int myDotPos, char
   myLookaheadSymbol);

  unsigned int lProductionRuleIdx;
  unsigned int rProductionRuleIdx;
  unsigned int dotPos;
  char lookaheadSymbol;

  friend bool operator < (const LR1Item& i1, const LR1Item& i2);
  friend bool operator ==(const LR1Item& i1, const LR1Item& i2);
  friend bool operator > (const LR1Item& i1, const LR1Item& i2);
  friend bool operator !=(const LR1Item& i1, const LR1Item& i2);

private:

};

class LR1ActionTabItem
{
public:
  LR1ActionTabItem() = default;
  LR1ActionTabItem(const Action& myAction, unsigned int
   myLProductionRuleIdx, unsigned int myRProductionRuleIdx,
   unsigned int myItemSetIdx);

```

```
  Action action;
  unsigned int lProductionRuleIdx;
  unsigned int rProductionRuleIdx;
  unsigned int itemSetIdx;

private:

};

vector<set<LR1Item>> lr1ItemSetFamily;
vector<vector<LR1ActionTabItem>> lr1ActionTab;
vector<vector<unsigned int>> lr1GoToTab;
vector<unsigned int> lr1Stack;
string lr1Input;
```

Listing 3: LR(1) 分析器数据结构

```
class LL1Item
{
public:
  LL1Item() = default;
  LL1Item(unsigned int myI, unsigned int myJ);

  unsigned int i;
  unsigned int j;

private:

};

vector<vector<LL1Item>> ll1ParseTab;
vector<char> ll1Stack;
string ll1Input;
```

Listing 4: LL(1) 分析器数据结构

9 附录: 算法

Algorithm 1 calFIRST(G)

```
1: repeat
2:   if X is a terminal symbol then
3:     FIRST(X) ← {X};
4:   else if X is a nonterminal symbol and X → Y_1 Y_2 ... Y_k ∈ G then
5:     for i from 1 to k do
6:       FIRST(X) ← FIRST(X) ⋃ FIRST(Y_i);
7:       if ε ∉ FIRST(Y_i) then
8:         break;
9:       else if i = k then
10:        FIRST(X) ← FIRST(X) ⋃ {ε};
11:      end if
12:    end for
13:  else if X → ε ∈ G then
14:    FIRST(X) ← FIRST(X) ⋃ {ε};
15:  end if
16: until no terminal symbols or ε were put into any FIRST
```

Algorithm 2 calFOLLOWG

```
1: repeat
2:   FOLLOW(S) ← FOLLOW(S) ⋃{$};
3:   if A → αBβ ∈ G then
4:     FOLLOW(B) ← FOLLOW(B) ⋃ (FIRST(β)-{ε});
5:     if β = ε or ε ∈ FIRST(β) then
6:       FOLLOW(B) ← FOLLOW(B) ⋃ FOLLOW(A);
7:     end if
8:   end if
9: until no terminal symbols were put into any FOLLOW
```

Algorithm 3 ItemSet closure(I)

```
1: J ← I;
2: repeat
3:    for each item A → α · Bβ ∈ J do
4:       for each production rule B → γ ∈ G do
5:          if item B → ·γ ∉ J then
6:             J ← J ⋃ {B → ·γ};
7:          end if
8:       end for
9:    end for
10: until no new items were put into J in the latest loop
11: return J;
```

Algorithm 4 ItemSet goto(I, X)

```
1: return closure({[A → αX · β] | [A → α · Xβ] ∈ I});
```

Algorithm 5 calItemSetFamily(G')

```
1: C ← { closure({[S' → ·S]})};
2: repeat
3:    for each item I ∈ C do
4:       for each grammar symbol X do
5:          if goto(I, X) ≠ ϕ and goto(I, X) ∉ C then
6:             C ← C ⋃ goto(I, X);
7:          end if
8:       end for
9:    end for
10: until no new item sets were put into C int the latest loop
```

Algorithm 6 construct an SLR(1) parse table

Input:

an expanded grammar G'

Output:

ACTION, GOTO

1: calItemSetFamily(G');
2: **if** $[A \to \alpha \cdot a\beta] \in I_i$ **and** GOTO(I_i, a)=I_j **and** a is a terminal symbol **then**
3: ACTION[i, a] $\leftarrow$ (SHIFT, j);
4: **else if** $[A \to \alpha\cdot] \in I_i$ **and** $A \neq S'$ **then**
5: **for all** $a \in$ FOLLOW(A) **do**
6: ACTION[i, a] $\leftarrow$ (REDUCE, $A \to \alpha$);
7: **end for**
8: **else if** $[S' \to S\cdot] \in I_i$ **then**
9: AXTION[i, a] $\leftarrow$ (ACCEPT, _);
10: **end if**
11: **if** GOTO[I_i, A]=I_j **then**
12: GOTO[i, A] $\leftarrow$ j;
13: **end if**
14: other items are set as (ERR, _);
15: initial state is the state constructed according to item set which contains $[S' \to \cdot S]$;

Algorithm 7 parse(ω, ACTION, GOTO)

```
1: push(stack, s_0);
2: buffer← ω$;
3: a ←first symbol of ω$;
4: while true do
5:    s ←stack[top];
6:    if ACTION[s, a]=(SHIFT, t) then
7:       push(stack, t);
8:       a ←next input symbol;
9:    else if ACTION[s, a]=(REDUCE, A → β) then
10:      for i ← 0; i<|β|; i ← i + 1 do
11:         pop(stack, _);
12:      end for
13:      t ←stack[top];
14:      push(stack, GOTO[t, A]);
15:      print  A → β;
16:   else if ACTION[s, a]=(ACCEPT, _) then
17:      break;
18:   else
19:      recover from error;
20:   end if
21: end while
```

Algorithm 8 ItemSet closure(I)

```
1: repeat
2:    for each item [A → α · Bβ, a] ∈ I do
3:       for each production rule B → γ ∈ G′ do
4:          for each terminal symbol b ∈ FIRST(βa) do
5:             I ← I ⋃{[B → ·γ, b]};
6:          end for
7:       end for
8:    end for
9: until no new items were put into I in the latest loop
10: return  I;
```

Algorithm 9 ItemSet goto(I, X)

1: $J \leftarrow \phi$;
2: **for** each item $[A \rightarrow \alpha \cdot X\beta, a] \in I$ **do**
3: $J \leftarrow J \bigcup \{[A \rightarrow \alpha X \cdot \beta, a]\}$;
4: **end for**
5: **return** closure(J);

Algorithm 10 calItemSetFamily(G')

1: $C \leftarrow \{$ closure($\{[S' \rightarrow \cdot S, \$]\}$)$\}$;
2: **repeat**
3: **for** each item $I \in C$ **do**
4: **for** each grammar symbol X **do**
5: **if** goto(I, X) $\neq \phi$ **and** goto(I, X) $\notin C$ **then**
6: $C \leftarrow C \bigcup$goto(I, X);
7: **end if**
8: **end for**
9: **end for**
10: **until** no new item sets were put into C int the latest loop

Algorithm 11 construct an LR(1) parse table

Input:
an expanded grammar G'
Output:
ACTION, GOTO
1: calItemSetFamily(G');
2: **if** $[A \rightarrow \alpha \cdot a\beta, b] \in I_i$ **and** GOTO(I_i, a)=I_j **and** a is a terminal symbol **then**
3: ACTION[i, a] $\leftarrow$ (SHIFT, j);
4: **else if** $[A \rightarrow \alpha \cdot, a] \in I_i$ **and** $A \neq S'$ **then**
5: ACTION[i, a] $\leftarrow$ (REDUCE, $A \rightarrow \alpha$);
6: **else if** $[S' \rightarrow S \cdot, \$] \in I_i$ **then**
7: AXTION[i, a] $\leftarrow$ (ACCEPT, _);
8: **end if**
9: **if** GOTO[I_i, A]=I_j **then**
10: GOTO[i, A] $\leftarrow$ j;
11: **end if**
12: other items are set as (ERR, _);
13: initial state is the state constructed according to item set which contains $[S' \rightarrow \cdot S, \$]$;

Algorithm 12 parse(ω, ACTION, GOTO)

```
1: push(stack, s_0);
2: buffer← ω$;
3: a ←first symbol of ω$;
4: while true do
5:    s ←stack[top];
6:    if ACTION[s, a]=(SHIFT, t) then
7:       push(stack, t);
8:       a ←next input symbol;
9:    else if ACTION[s, a]=(REDUCE, A → β) then
10:      for i ← 0; i<|β|; i ← i + 1 do
11:         pop(stack, _);
12:      end for
13:      t ←stack[top];
14:      push(stack, GOTO[t, A]);
15:      print  A → β;
16:   else if ACTION[s, a]=(ACCEPT, _) then
17:      break;
18:   else
19:      recover from error;
20:   end if
21: end while
```

Algorithm 13 construct an LL(1) parse table

Input:
a grammar G
Output:
M

for each production rule $A \to \alpha \in G$ **do**
  for each terminal symbol $a \in \text{FIRST}(\alpha)$ **do**
    $M[A, a] \leftarrow M[A, a] \bigcup \{A \to \alpha\}$;
  end for
  if $\varepsilon \in \text{FIRST}(\alpha)$ **then**
    for each terminal symbol $b \in \text{FOLLOW}(A)$ **do**
      $M[A, b] \leftarrow M[A, b] \bigcup \{A \to \alpha\}$;
    end for
    if \$ $\in \text{FOLLOW}(A)$ **then**
      $M[A, \$] \leftarrow M[A, \$] \bigcup \{A \to \alpha\}$;
    end if
  end if
end for

Algorithm 14 parse(ω, M)

ip points to the first symbol of ω;
$X \leftarrow$ stack[top];
while $X \neq$ \$ **do**
  if $X = a$ which ip points to **then**
    pop(stack, _);
    make ip move forward a step;
  else if X is a terminal symbol **then**
    err;
  else if $M[X, a]$=err **then**
    err;
  else if $M[X, a] = X \to Y_1 Y_2 \dots Y_k$ **then**
    print $X \to Y_1 Y_2 \dots Y_k$;
    pop(stack, _);
    for i from k to 1 **do**
      push(stack, Y_i);
    end for
  end if
  $X \leftarrow$ stack[top];
end while