Lecture 8: 二义性和错误恢复

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

8.1 二义性

二义性文法的自底向上分析

- **二义性文法都不是LR的**
- **二义性文法却有其存在的必要**
- **对于某些二义性文法**
  - 可以通过消除二义性规则来保证每个句子只有一棵语法分析树
  - 且可以在LR分析器中实现这个规则

优先级/结合性消除冲突

$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \mathbf{id}$$

- **二义性文法的优点**
  - 容易修改算符的优先级和结合性
  - 简洁：较少的非终结文法符号
  - 高效：不需要处理E → T这样的归约

二义性表达式文法的LR(0)项集

$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

I_7，I_8中有冲突，在输入+或*时，不能确定是归约还是移入，且不可能通过向前看符号解决

I_0:
$E' \rightarrow \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_1:
$E' \rightarrow E \cdot$
$E \rightarrow E \cdot + E$
$E \rightarrow E \cdot * E$

I_2:
$E \rightarrow (\cdot E)$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_3:
$E \rightarrow \mathbf{id} \cdot$

I_4:
$E \rightarrow E + \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_5:
$E \rightarrow E * \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_6:
$E \rightarrow (E \cdot)$
$E \rightarrow E \cdot + E$
$E \rightarrow E \cdot * E$

I_7:
$E \rightarrow E + E \cdot$
$E \rightarrow E \cdot + E$
$E \rightarrow E \cdot * E$

I_8:
$E \rightarrow E * E \cdot$
$E \rightarrow E \cdot + E$
$E \rightarrow E \cdot * E$

I_9:
$E \rightarrow (E) \cdot$

基于优先级解决冲突

前缀	栈	输入
$E + E$	0 1 4 7	$*$ **id** $

- **设定优先级如下：*的优先级大于+，且+是左结合的，则有**
  - 下一个符号为+时，我们应该将E+E归约为E
  - 下一个符号为*时，我们应该移入*，期待移入下一个符号

解决冲突之后的SLR(1)分析表

- **对于状态7，输入**
  - +时归约
  - *时移入
- **对于状态8**
  - 执行归约

I_7: $E \to E + E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

I_8: $E \to E * E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

状态	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

悬空else的二义性

$S' \rightarrow S$

$S \rightarrow i\ S\ e\ S \mid i\ S \mid a$

- 栈中内容 if *expr* then *stmt*,

是输入else，还是归约？

- 答案是移入

Follow(S)={e,$}

I_0:
$S' \rightarrow \cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_1:
$S' \rightarrow S\cdot$

I_2:
$S \rightarrow i\cdot SeS$
$S \rightarrow i\cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_3:
$S \rightarrow a\cdot$

I_4:
$S \rightarrow iS\cdot eS$
$S \rightarrow iS\cdot$

I_5:
$S \rightarrow iSe\cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_6:
$S \rightarrow iSeS\cdot$

8.2 Error Handling

语法错误的处理

- **错误难以避免**
- **编译器需要具有处理错误的能力**
- **程序中可能存在不同层次的错误**
  - 词法错误；语法错误；语义错误；逻辑错误
- **语法错误相同容易发现，语义和逻辑错误较难精确的检测到**
- **语法分析器中错误处理程序的设计目标**
  - 清晰准确地**报告**出现错误，并指出错误的**位置**
  - 能从当前错误中**恢复**，以继续检测后面的错误
  - 尽可能减少处理正确程序的开销

预测分析中的错误恢复

- **错误恢复**
  - 当预测分析器报错时，表示输入的串不是句子
  - 对于使用者而言，希望预测分析器能够进行恢复处理后继续语法分析过程，以便在一次分析中找到更多的语法错误
  - 但是有可能恢复得并不成功，之后找到的语法错误有可能是假的
  - 进行错误恢复时可用的信息：栈里面的符号，待分析的符号
- **两类错误恢复方法**
  - 恐慌模式
  - 短语层次的恢复

基本思想

- 分析表对应的 Action 空白项表示出错，
- 一般处理方法：
  - 如果出错状态含有项目 $A \to \alpha \bullet \beta$，表示希望形成形如``$\alpha\beta$''的句柄，但是当前的错误输入使得分析器不能移进，此时可采用类似 LL 分析的方法，跳过当前的输入直到非终结符 A 的同步符号出现，用 $A \to \alpha\beta$ 归约继续分析.
  - 如果出错状态含有项目 $A \to \alpha\bullet$，表示当前的输入不是 A 的 Follow 集元素，处理方法，归约该项目，转入和前者一样的处理.

例子

- **E→E+E, E→ E*E, E→(E), E→ id**

	Action						Goto
状态	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r2		r3	r3	

I_0:
$E' \to \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_1:
$E' \to E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_2:
$E \to (\cdot E)$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_3:
$E \to \mathbf{id}\cdot$

I_4:
$E \to E + \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_5:
$E \to E * \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_6:
$E \to (E\cdot)$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_7:
$E \to E + E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_8:
$E \to E * E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_9:
$E \to (E)\cdot$

例子

e1：这个例程在状态0、2、4和5上被调用。所有这些状态都期望读入一个运算分量的第一个符号，这个符号可能是id或左括号，但是实际读入的却是+、*或输入结束标记。

将状态3(状态0、2、4和5在输入id上的GOTO目标)压入栈中；

发出诊断信息“缺少运算分量。”

e2：在状态0、1、2、4和5上发现输入为右括号时调用这个过程。

从输入中删除右括号；

发出诊断信息“不匹配的右括号。”

e3：当在状态1和6上，期待读入一个运算符却发现了一个id或左括号时调用。

将状态4(对应于符号+的状态)压入栈中。

发出诊断信息“缺少运算符。”

e4：当在状态6上发现输入结束标记时调用。

将状态9(对应于右括号)压入栈中；

发出诊断信息“缺少右括号。”

在处理错误的输入id +)时，语法分析器进入的格局序列显示在图4-54中。 □

4.8.4 4.8节的练习

	Action						Goto
状态	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r2		r3	r3	

I_0: $E' \to \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_1: $E' \to E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_2: $E \to (\cdot E)$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_3: $E \to \mathbf{id}\cdot$

I_4: $E \to E + \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_5: $E \to E * \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_6: $E \to (E\cdot)$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_7: $E \to E + E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_8: $E \to E * E\cdot$
$E \to E\cdot + E$
$E \to E\cdot * E$

I_9: $E \to (E)\cdot$

例子

分析表的出错处理

1/ E → E+E　2/ E → E*E　3/ E → (E)　4/ E → id

	Action						Goto
状态	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r2	r3	r3	r3	

例子

- **分析过程**

作业

- **教材p182：4.8.4**

8.2 YACC

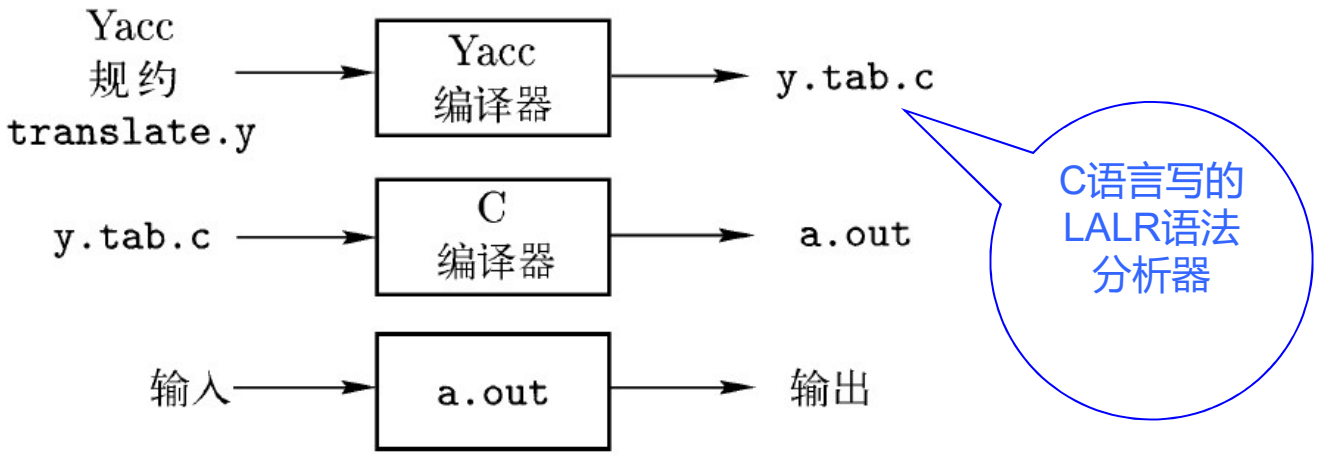
自下而上语法分析器生成工具

Examples

1. yacc(Yet Another Compiler-Compiler): 1975 年由贝尔实验室 Mike Lesk & Eric Schmidt 开发，UNIX 标准实用工具 (utility);
2. byacc: Berkeley YACC: Robert Corbett, 1989 年, yacc compatible, in Free BSD distribution, DOS version in my CD-ROM;
3. bison: Robert Corbett & Richard Stallmen, 1988 年，yacc compatible, in Linux distribution, 最新版本：2.4. 支持 GLR(Generalized LR) 文法，http://www.gnu.org/software/bison/.
4. CUP: LALR Parser Generator in Java, current version 0.11a, 对应的词法分析器生成工具为：JFLex(http://jflex.de/), http://www2.cs.tum.edu/projects/cup/.
5. A. Holub LRpars: See CD-ROM, 支持动态显示分析过程.

语法分析器生成工具YACC

- **YACC的使用方法如下：**



YACC源程序的结构

- **声明**
  - 分为可选的两节：第一节放置C声明，第二节是对词法单元的声明。
- **翻译规则：**
  - 指明产生式及相关的语义动作
- **辅助性C语言例程**
  - 被直接拷贝到生成的C语言源程序中，
  - 可以在语义动作中调用。
  - 其中必须包括yylex()。这个函数返回词法单元，可以由LEX生成

声明
%%
翻译规则
%%
辅助性C语言程序

翻译规则的格式

<产生式头>：　　<产生式体>1 {<语义动作>1}
　　|　　<产生式体>2 {<语义动作>2}
　　… … … …
　　|　　<产生式体>n {<语义动作>n}
　　;

- 第一个产生式的头被看作开始符号；
- 语义动作是C语句序列；
- $$表示和产生式头相关的属性值，$i表示产生式体中第i个文法符号的属性值。
- 当我们按照某个产生式归约时，执行相应的语义动作。通常可以根据$i来计算$$的值。
- 在YACC源程序中，可以通过定义YYSTYPE来定义$$，$i的类型。

YACC源程序的例子

```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line   : expr '\n'        { printf("%d\n", $1); }
       ;
expr   : expr '+' term    { $$ = $1 + $3; }
       | term
       ;
term   : term '*' factor  { $$ = $1 * $3; }
       | factor
       ;
factor : '(' expr ')'     { $$ = $2; }
       | DIGIT
       ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

YACC对于二义性文法的处理

- **缺省的处理方法**
  - 对于归约/归约冲突，选择前面的产生式
  - 对于归约/移入冲突，总是移入（悬空-else的解决）
- **运行选项-v可以在文件y.output中看到冲突的描述及其解决方法；**
- **可以通过一些命令来确定终结符号的优先级/结合性，解决移入/归约冲突。**
  - 结合性：%left　　%right　　%nonassoc
  - 终结符号的优先级通过它们在声明部分的出现顺序而定。
  - 产生式的优先级设定为它的最右终结符号的优先级。也可以加标记%prec<终结符号>，指明产生式的优先级等同于该终结符号
  - 移入符号a/按照A→α归约：当A→α的优先级高于a，或者两者优先级相同但产生式是左结合时，选择归约，否则移入。

YACC的错误恢复

- **使用错误产生式的方式来完成语法错误恢复**
  - 错误产生式A → **error** α
  - 例如：*stmt* → **error** ;
- **首先定义哪些“主要”非终结符号有相关的错误恢复动作；**
  - 比如：表达式，语句，块，函数定义等对应的非终结符号
- **当语法分析器碰到错误时**
  - 不断弹出栈中状态，直到有一个状态包含A →.**error** α;
  - 分析器将error移入栈中
  - 如果α为空，分析器直接执行归约，并调用相关的语义动作；否则向前跳过一些符号，直到找到可以归约为α的串

作业

- **课程设计1：**
  - 11月26日18:00 前提交，超过一天扣分10%

Thank you!