编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第五章 语法分析
-自下而上分析

第五章 语法分析-自下而上分析

- 本章目录

第五章 语法分析-自下而上分析

- 大纲要求
- 掌握：归约，规范归约，算法优先分析法（算符优先文法、优先表的构造、算符优先分析算法、优先函数的构造）。
- 理解：符号栈的使用方法。
- 了解：自下而上语法分析的基本原理和工作方法，语法分析器的自动产生工具YACC的基本作用。

5.3 LR分析法

5.3 LR分析法

- 5.3.1 LR分析器
- 5.3.2 LR(0)分析
- 5.3.3 SLR(1)分析
- 5.3.4 LR(1)分析
- 5.3.5 LALR(1)分析

5.3.1 LR分析器

5.3.1 LR分析器

- LR分析器的概念
- Left-to-right Right-most

  L Left-to-right 从左到右扫描输入字符串

  R Right-most 最左推导

5.3.1 LR分析器

- LR分析法的特点
- LR分析器（程序）基本上可以识别所有上下文无关文 法写的编程语言结构，分析能力强且适用范围广。
- LR分析法是当前最一般,移进-归约分析方法。
- LR分析法在自左向右扫描输入串时能发现其中错误， 并能指出出错地点。
- LR分析程序可以自动生成。

5.3.1 LR分析器

- LR分析器的概念
- 基本思想
- 在规范归约的过程中，一方面记住已移进和归约出的整个符号串，即记住“历史”，另一方面根据所用的产生式推测未来可能碰到的输入符号，即对未来进行“展望”。

历史容易获得，但展望难以得到，推测未来的一个符号，右非常多的不同可能性

5.3.1 LR分析器

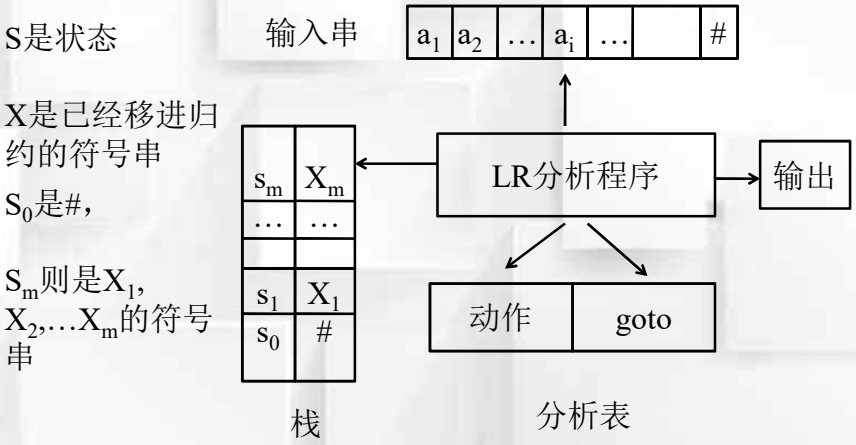
- LR分析器的概念

S是状态

X是已经移进归约的符号串

S_0是#，

S_m则是X_1, X_2,…X_m的符号串



5.3.1 LR分析器

- LR分析器的概念
- 核心是一张分析表，包括两个部分，

  一是动作ACTION[s, a]，另一是状态转换GOTO[s, X]。

  ACTION[s, a]规定当状态s面临输入符号a时应采取什么动作。

  GOTO[s, X]规定了状态s面对文法符号X(终结符或非终结符)时下一状态是什么。

LR分析表

文法G(E)：
(1) E→E+T | T
(2) T→T*F | F
(3) F→P↑F | P
(4) P→(E) | i

状态	ACTION(动作)				GOTO(转换)		
	i	+	*	#	E	T	F
0	s5				1	2	3
1		s6		acc			
2		r2	s7	r2			
3		r4	r4	r4			
4	s5				8	2	3

s5代表移进当前符号和移进状态5
r2代表使用文法G的第2个产生式归约
acc是接受
空白是出错标志，报错

13

LR分析表

状态	ACTION(动作)				GOTO(转换)		
	i	+	*	#	E	T	F
0	s5				1	2	3
1		s6		acc			
2		r2	s7	r2			
3		r4	r4	r4			
4	s5				8	2	3

s5代表移进当前符号与状态5
r2代表使用文法G的第2个产生式归约
acc是接受
空白是出错标志，报错

	状态	符号	输入串
(1)	0	#	i*i+i#
(2)	05	#i	*i+i#
(3)	03	#F	*i+i#
(4)	02	#T	*i+i#
(5)	027	#T*	i+i#
(6)	0275	#T*i	+i#

14

如何建立LR文法
使用的分析表？

5.3.1 LR分析器

- LR分析表的种类
- LR（0）：最简单分析表，局限性大，是其它分析表的基础。
- SLR： 简单分析表，容易实现，功能比LR（0）稍强些。
- LR（K）：分析能力最强，但实现代价高。
- LALR分析表：称为向前看LR分析表，功能介于SLR（1）和 LR（1）之间，适用于大多数程序设计语言的结构，并且可以 比较有效地实现

5.3.2 LR(0)分析

5.3.2 LR(0)分析

- LR(0)

  可以直接从它的**项目集规范族**和**活前缀识别自动机的状态转换函数GO**构造出LR分析表。

项目集规范族

5.3.2 LR(0)分析

- LR(0)
- 例如：符号栈有字符#T，下一个输入符号是*，
- 考虑下面的文法G(E)：

E→E+T | T
T→T*F | F
F→P↑F | P
P→(E) | i

难以确定是移进*，还是归约T到E

5.3.2 LR(0)分析

- 文法G的LR(0)项目

  在G的产生式中加入点"."

- 例如：$A \to X\,Y\,Z$ 有如下的4个LR(0)项目

  $A \to .X\,Y\,Z$

  $A \to X\,.Y\,Z$

  $A \to X\,Y\,.Z$

  $A \to X\,Y\,Z\,.$

  $A \to \varepsilon$ 只有1个项目$A \to .$

5.3.2 LR(0)分析

- 文法G的LR(0)项目

  $A \to .X\,Y\,Z$ 表示**希望看到**一个紧跟输入字符"."的字符串，该字符串由XYZ导出

  $A \to X\,.Y\,Z$
  $A \to X\,Y\,.Z$ 表示**已经看到**一个输入字符串由X或XY导出，**并希望看到**紧跟一个字符串，该字符串由YZ或Z导出

  $A \to X\,Y\,Z\,.$ 表示**已经看到**一个输入字符串由XYZ导出，并且应该把XYZ**归约**到A

5.3.2 LR(0)分析

- LR(0)

  圆点在最右端的项目， A→ a .，称为一个“**归约项目**”

  对文法开始符号S’的归约项目， S’ → a .，称为“**接受**”

  A→ α **.a** β的项目，其中a为终结符，称为“**移进**”项目

  A→ α **.B** β的项目，其中B为非终结符，称为“待约”项目

5.3.2 LR(0)分析

- 文法G的LR(0)项目

  一组这样的LR(0)项目，叫做LR(0)项目集族。这样的集族可以提供建立DFA的基本信息，以便于做出语法分析的判定。

  通过对所有项目集形成的闭包CLOSURE(I) 就是项目集规范族。

5.3.2 LR(0)分析

- LR(0)项目闭包CLOSURE

  如果I是文法G的一组LR(0)项目集族，CLOSURE(I)由两个规则组成：

a. 把所有I中的项目加入CLOSURE(I)

b. 如果A→α **.B** β在CLOSURE(I)中，并且**B**→γ，如果**B**→**.**γ不存在于CLOSURE(I)中，则把**B**→**.**γ加入CLOSURE(I)。重复此项操作，直到没有新加项为止。

闭包只对输入为终结符，或初始I中的项目

5.3.2 LR(0)分析

- 文法G的LR(0)分析
  - 考虑下面的文法G(E)：

    E→E+T | T
    T→T*F | F
    F→(E) | i

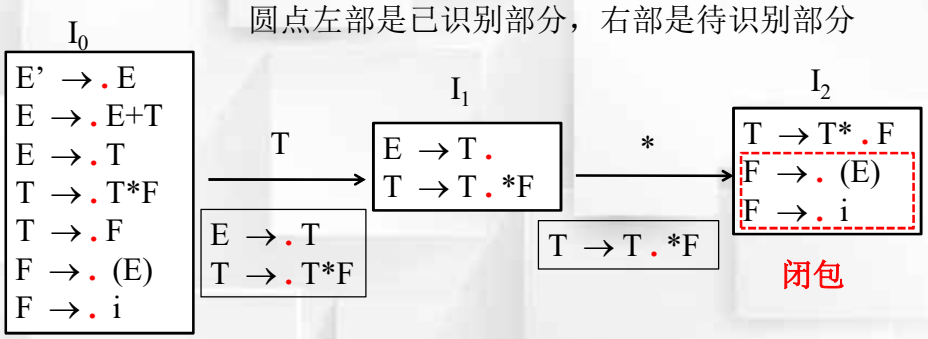
I是{[E’ → . E]}, 则CLOSURE(I)是什么？

E’ → . E
E → . E+T
E → . T
T → . T*F
T → . F
F → . (E)
F → . i

- 考虑下面的文法G(E)：符号栈有字符#T，下一个输入符号是*，是移进*，还是归约T到E？

  E→E+T | T
  T→T*F | F
  F→(E) | i

圆点左部是已识别部分，右部是待识别部分



活前缀自动识别机的状态转换函数

GO

5.3.2 LR(0)分析

- LR(0)

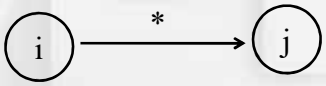
要识别文法G的所有**活前缀，需要构造NFA**

1. 列出文法G的所有项目，并标记数字。并设定文法G的第一个产生式的第一个项目为文法的**初态。**

2. 如果状态i和状态j出自同一个产生式，而且状态j的圆点只落后于状态i的一个为止，则画一条箭弧从i到j，并标记Xi(当前状态已移进归约)

例如：状态i为T → T . *F

状态j为T → T* . F



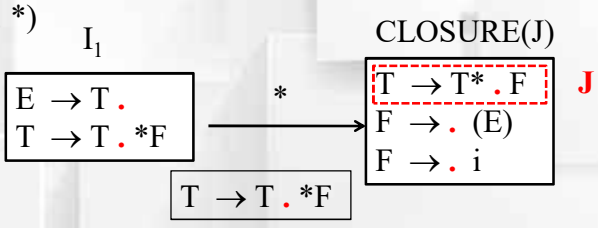
5.3.2 LR(0)分析

- LR(0)

函数GO是状态转换函数。GO(I, X)中的I是一个项目集，X是一个文法符号，J也是一个项目集。

GO(I, X) = CLOSURE(J)

例如： GO(I_1, *)



5.3.2 LR(0)分析

- 拓广文法G’

  有一个文法G，它的拓广为G’。

  G’包含了整个G，另外引进一个非终结符S’为文法G’的开始符号，G的开始符号为S，S’→S。

  拓广的意义：让“接受(acc)”状态易于识别。

  S’→ S . 属于某个项目集规范族I，则把分析表中的ACTION[I, #]设为接受。

5.3.3 SLR(1)分析

5.3.3 SLR(1)分析

- SLR(1)
  - 例子： I = {X → α .b β, 移进
    A → α ., α属于FOLLOW(A), 归约
    B → α .} α属于FOLLOW(B), 归约

两个归约，引起冲突性动作

如果FOLLOW(A)和FOLLOW(B)没有交集，而且都不包含b，则可以避免冲突

5.3.3 SLR(1)分析

- SLR(1)
  - 例子： I = {X → α .b β, 移进
    A → α ., α属于FOLLOW(A), 归约
    B → α .} α属于FOLLOW(B), 归约

SLR(1)是一种解决冲突性动作的办法，叫做**SLR(1)解决办法，Simple LR分析法，简单LR分析法。**

I中有m个移进项目，和n个归约项目，因此可以通过检查现行输入符号a属于哪个集合而获得解决。

5.3.4 LR(1)分析

5.3.4 LR(1)分析

- 活前缀Viable Prefixes

  活前缀：规范句型的一个前缀，前缀不包含句柄之后的任何符号。

  $E \underset{rm}{\overset{*}{\Rightarrow}} F * I \Rightarrow (E) * i$

- 在分析时，符号栈会有(, (E, 和 (E)，但基本不会有(E)*，因为(E)是一个句柄，会把(E)归约为F，而不是移进*。(E) 是一个活前缀。

5.3.4 LR(1)分析

- LR(1)

  把项目加入一个终结符，该终结符作为项目的第二个组成。

  一个项目的通用形式变成：[A → α . β, a]
  其中A → α β是一个产生式，a是一个终结符或右结尾符号#
  这样的一个形式叫做**LR(1)项**。

  1 是右边这个终结符的长度，叫做当前项的**Lookahead**(字符串长度可以大于1)。当前长度为1，即向前搜索1个字符。

5.3.4 LR(1)分析

- LR(1)

  如果$S \overset{*}{\underset{rm}{\Rightarrow}} \sigma A w \overset{*}{\underset{rm}{\Rightarrow}} \sigma \alpha \beta w$，并且$\gamma = \sigma \alpha$, a是w中的第一个字符，或当w是ε时a是结尾符号#。 $\sigma \alpha \beta w = \gamma \beta w$

  我们说LR(1)是项[A → α . β, a]对于活前缀γ是有效的

5.3.5 LALR(1)分析

5.3.5 LALR(1)分析

- LALR(1) (LookAhead LR)

  这是实际中经常用到的分析方法，由该方法建立的分析表比LR分析表更小。

  该方法根SLR分析相似，只是在建表过程中，让某些步骤更简便。

5.3.5 LALR(1)分析

- SLR vs LALR vs LR

  对于同一个语法，SLR和LALR建立的分析表由相同数量的状态。如果是对C语言建立分析表，这个状态的数量有数百个。

  如果用LR建立C的分析表，状态数量则有数千个。

5.4 语法分析器的自动产生工具YACC

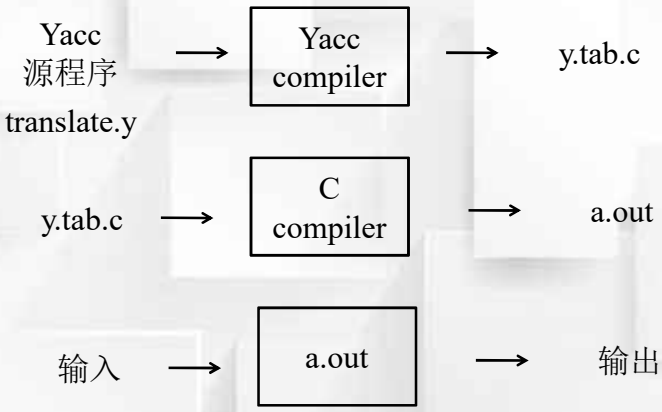
5.4语法分析器的自动产生工具YACC

- 语法分析器的自动产生工具YACC
  Yet Another Compiler-Compiler

  这个工具是1970年由S. C. Johnson开发，并且是很流行的分析程序的生成工具。

5.4语法分析器的自动产生工具YACC

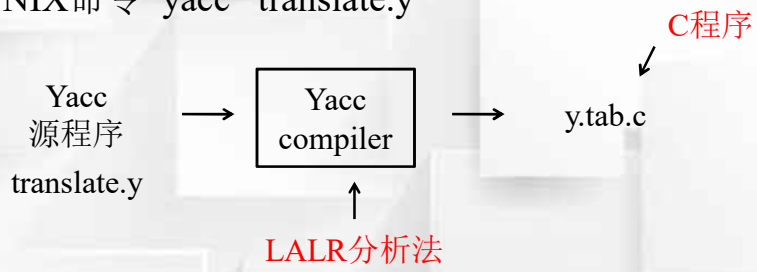
- 使用YACC生成输入\输出的编译程序的过程



5.4语法分析器的自动产生工具YACC

- 使用YACC生成输入\输出的编译程序的过程

UNIX命令 yacc translate.y



45

5.4语法分析器的自动产生工具YACC

- 使用YACC生成输入\输出的编译程序的过程

UNIX命令 yacc translate.y

Yacc
源程序
translate.y

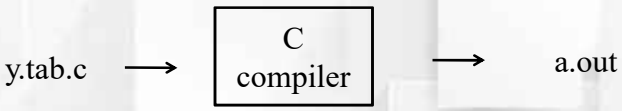
Yacc源程序包含3个部分：
声明(declarations)
%%
规则(translation rules)
%%
程序(supporting C routines)

46

5.4语法分析器的自动产生工具YACC

- 使用YACC生成输入\输出的编译程序的过程

UNIX命令 cc y.tab.c -ly ← ly库，含有LR分析程序



47

5.4语法分析器的自动产生工具YACC

- YACC的声明部分

定义语法规则中要用的终结符号，语义动作中使用的数据类型、 变量、语义值的联合类型以及语法规则中运 算符的优先级等。声明部分可以是空的。

包含两个部分，C语言的声明和语法的标识符token

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

要使用的头文件，包含功能isdigit()

```
% token DIGIT

%%
```

48

5.4语法分析器的自动产生工具YACC

- YACC的规则部分

  在第一组%%之后

  语法规则部分是整个YACC源程序的主体，它是由一组产生式及相应的语义动作组成。

  <head> → <body>$_1$ | <body>$_2$ |…| <body>$_n$ |

  Yacc中写成如下形式

  <head> → <body>$_1$ {<语义动作>$_1$}

  <body>$_2$ {<语义动作>$_2$}

  …

  <body>$_n$ {<语义动作>$_n$}

49

5.4语法分析器的自动产生工具YACC

- YACC的规则部分

  在此部分，所有没有用引号括起来的字符串和没有在声明为token的数字，都是非终结符。例如，‘c’被认为是终结符c。

  <head> → <body>$_1$ {<语义动作>$_1$}

  expr : expr ‘+’ term {$$ = $1 + $3; } $E \rightarrow E + T \mid T$

  | term

  ;

  $$对应expr, $1对应产生式右部的第一个语法符号expr, + 对应+, $3对应产生式右部第三个语法符号term。

  语义动作为执行加法，并把值赋值给expr。

50

5.4语法分析器的自动产生工具YACC

- YACC的规则部分

```
<head>  → <body>1          {<语义动作>1}
  line    :  expr  '\n'     {printf("%d\n", $1);}
           ;
  expr    :  expr  '+'  term   {$$ = $1 + $3;  }
           ;
  term    :  term  '*'  factor {$$ = $1 * $3;  }
           ;
  factor   :  '('  expr  ')'   {$$ = $2;  }
         |  DIGIT
           ;
  %%
```


5.4语法分析器的自动产生工具YACC

- YACC的程序部分

```
yylex(){
    int c;
    c = getchar();
    if ( isdigit(c) ) {
       yylval = c – '0';
       return DIGIT;
    }
    return c;
}
```

yylex()词法分析函数, 必须给。通常使用Lex词法分析程序生成yylex()。

DIGIT是定义的标识符token，yylval (Yacc定义的变量，用于传递定义的token的属性值) 是DIGIT对应的属性值。

注意这里的c对应的字符和'0'都是C语言中的字符码。

5.4语法分析器的自动产生工具YACC

- YACC对冲突的处理

YACC生成LALR(1)分析器，如果接受的文法不是LALR(1)分析表就有冲突。

YACC解决冲突的默认规则为：

① 归约—归约冲突：选择YACC源程序中排列 在前面的产生式进行归约；

② 移进—归约冲突：移进动作优先于归约动 作。

第五章 小结

第五章 小结

5.1自下而上分析基本问题

5.2算符优先分析

5.3 LR分析法

5.4语法分析器的自动产生工具YACC

Coursework

5.1令文法G_1为

E→E+T｜T

T→T*F｜F

F→(E)｜i

证明E+T*F是它的一个句型，指出这个句型的所有短语，直接短语和句柄。

Coursework

5.2考虑下面的表格结构文法G_2：

S→a｜∧｜(T)

T→T，S｜S

（1）给出(a，(a，a))和(((a，a)，∧，(a))，a)的最左和最右推导。

（2）指出(((a，a)，∧，(a))，a)的规范归约及每一步的句柄。根据这个规范归约，给出“移进-归约”的过程，并给出它的语法树自下而上的构造过程。

Coursework

5.3（1）计算练习5.2文法G_2的FIRSTVT和LASTVT。

（2）计算G_2的优先关系。G_2是一个算符优先文法吗？

（3）计算G_2的优先函数。

（4）给出输入串(a，(a，a)) 的算符优先分析过程。