编译原理

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

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR(0) 项目集规范族
  - LR(0) 分析表的构造
  - SLR 分析表的构造
  - LR(1) 分析表的构造

SLR 冲突消解存在的问题

- 计算 FOLLOW 集合所得到的超前符号集合可能大于实际能出现的超前符号集。

$$FOLLOW(A) = \{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T\}$$

- 非 SLR 文法示例：
  - (0) S′→S
  - (1) S→L=R
  - (2) S→R
  - (3) L→*R
  - (4) L→i
  - (5) R→L

这个文法的 LR(0) 项目集规范族为

I_0 : $S' \rightarrow \cdot S$

$S \rightarrow \cdot L=R$

$S \rightarrow \cdot R$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

$R \rightarrow \cdot L$

I_1 : $S' \rightarrow S \cdot$

I_2 : $S \rightarrow L \cdot = R$

$R \rightarrow L \cdot$

I_3 : $S \rightarrow R \cdot$

I_4 : $L \rightarrow * \cdot R$

$R \rightarrow \cdot L$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

I_5 : $L \rightarrow i \cdot$

I_6 : $S \rightarrow L= \cdot R$

$R \rightarrow \cdot L$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

I_7 : $L \rightarrow *R \cdot$

I_8 : $R \rightarrow L \cdot$

I_9 : $S \rightarrow L=R \cdot$

I_2：

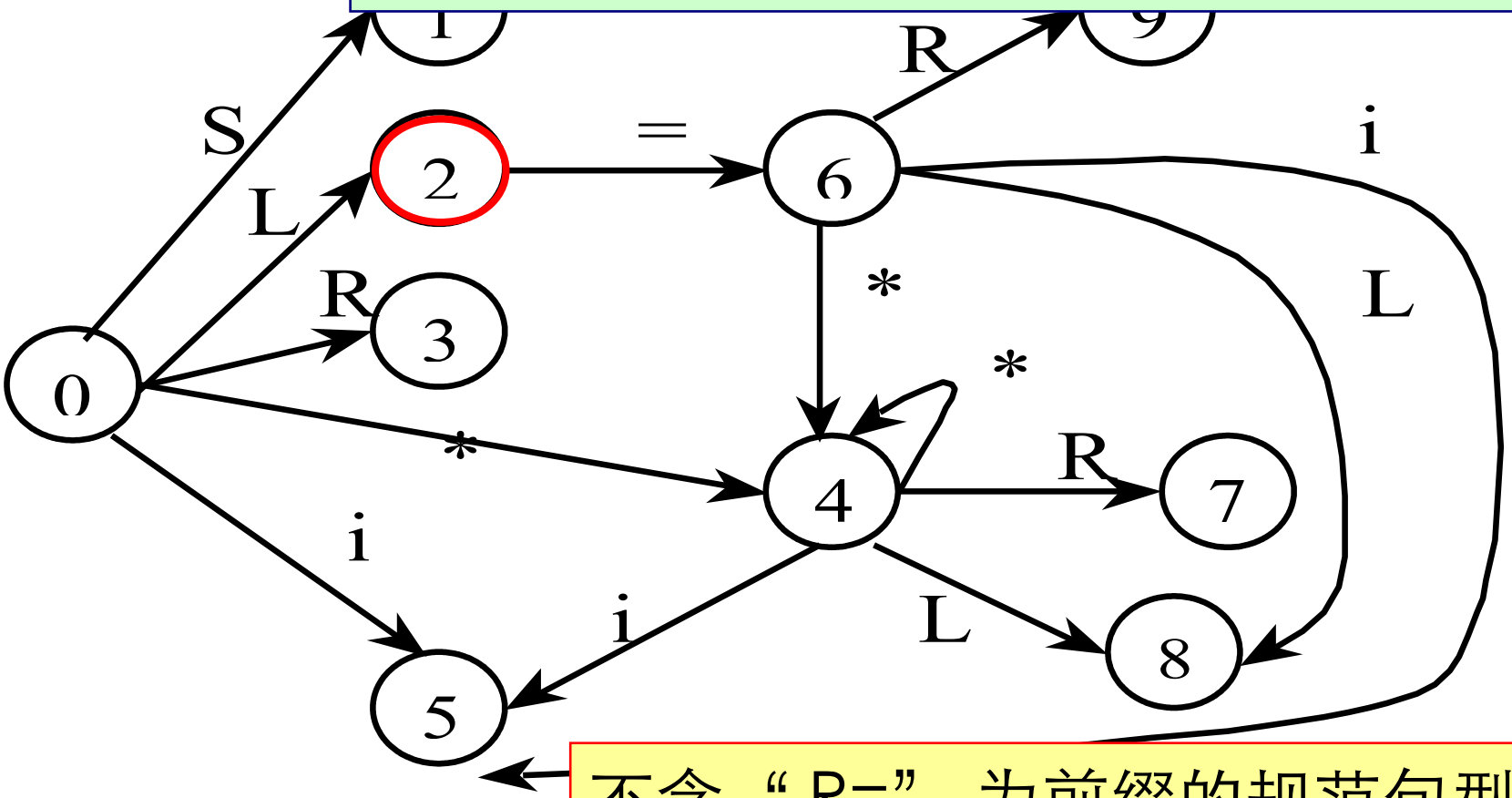
$S \rightarrow L \cdot = R$

$R \rightarrow L \cdot$

含有“移进 – 归约”冲突

FOLLOW(R) = {#, =}

当状态 2 显现于栈顶而且面临输入符号为‘ =’ 时，实际上不能用对栈顶 L 进行归约。



不含“ R=” 为前缀的规范句型

有含“ *R=” 为前缀的规范句型

SLR 冲突消解存在的问题

- SLR 在方法中，如果项目集 I_i 含项目 $A\rightarrow\alpha\cdot$ 而且下一输入符号 $a\in FOLLOW(A)$，则状态 i 面临 a 时，可选用“用 $A\rightarrow\alpha$ 归约”动作
- 但在有些情况下，当状态 i 显现于栈顶时，栈里的活前缀未必允许把α归约为 A，因为可能根本就不存在一个形如“β Aa” 的规范句型
- 在这种情况下，用“ $A\rightarrow\alpha$ ” 归约不一定合适
- FOLLOW 集合提供的信息太泛!

$$FOLLOW(A)=\{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a\in V_T\}$$

5.3.4 规范 LR 分析表的构造

- 我们需要重新定义项目，使得每个项目都附带有 k 个终结符
- 每个项目的一般形式是

$$[A\rightarrow\alpha\cdot\beta, a_1a_2\cdots a_k]$$

这样的一个项目称为一个 LR(k) 项目。项目中的 $a_1a_2\cdots a_k$ 称为它的向前搜索符串 (或展望串)。

- 向前搜索符串仅对归约项目 $[A\rightarrow\alpha\cdot\ ,\ a_1a_2\cdots a_k]$ 有意义
- 对于任何移进或待约项目 $[A\rightarrow\alpha\cdot\beta, a_1a_2\cdots a_k]$, $\beta\neq\varepsilon$，搜索符串 $a_1a_2\cdots a_k$ 没有直接作用

归约项目

- 归约项目 $[A\rightarrow\alpha\cdot, a_1a_2\cdots a_k]$ 意味着：当它所属的状态呈现在栈顶且后续的 k 个输入符号为 $a_1a_2\cdots a_k$ 时，才可以把栈顶上的α归约为 A
- 我们只对 $k\leq 1$ 的情形感兴趣，向前搜索 (展望) 一个符号就多半可以确定“移进”或“归约”

有效项目

- 形式上我们说一个 LR(1) 项目 [A→α · β, a] 对于活前缀γ是有效的，如果存在规范推导

$$S \underset{R}{\overset{*}{\Rightarrow}} \delta A \omega \underset{R}{\Rightarrow} \delta \alpha \beta \omega$$

其中， 1) γ = δα； 2) a 是ω的 第一个符号，或者 a 为 # 而ω为ε。

LR(1) 项目集规范族

- 为构造有效的 LR(1) 项目集规范族我们需要两个函数 CLOSURE 和 GO 。

- [A→α·Bβ, *a*] 对活前缀γ = δα是有效的，则对于每个形如 B→ξ 的产生式， 对任何 b∈FIRST(β*a*) ， [B→·ξ, b] 对γ也是有效的。

- 证明：若项目 [A→α·Bβ, *a*] 对γ = δα有效，则有规范推导

$$S \underset{R}{\overset{*}{\Rightarrow}} \delta Aa\chi \underset{R}{\Rightarrow} \delta\alpha B\beta a\chi \qquad \gamma = \delta\alpha$$

$$\because\ b \in FIRST(\beta a) \qquad \therefore \beta a\chi \underset{R}{\overset{*}{\Rightarrow}} b\omega$$

若$B \to \xi$是产生式

则$S \underset{R}{\overset{*}{\Rightarrow}} \delta\alpha B\beta a\chi \underset{R}{\Rightarrow} \gamma Bb\omega \underset{R}{\Rightarrow} \gamma\xi b\omega$

$\therefore$ 项目$[B \to .\xi, b]$对于γ是有效的

项目集的闭包 CLOSURE(I)

1. I 的任何项目都属于 CLOSURE(I) 。

2. 若项目 [A→α · Bβ, *a*] 属于 CLOSURE(I) ， B→ξ 是一个产生式，那么，对于 FIRST(β*a*) 中的每个终结符 *b* ，如果 [B→ · ξ, *b*] 原来不在 CLOSURE(I) 中，则把它加进去。

3. 重复执行步骤 2 ，直至 CLOSURE(I) 不再增大为止。

项目集的转换函数 GO

- 令 I 是一个项目集， X 是一个文法符号，函数 GO(I ， X) 定义为：

$$GO(I, X) = CLOSURE(J)$$

其中

$$J = \{ \text{任何形如 } [A \to \alpha X \cdot \beta, a] \text{ 的项目} \mid [A \to \alpha \cdot X\beta, a] \in I \}$$

LR(1) 项目集规范族的构造算法

- 计算 LR(1) 项目集规范族 C

```
BEGIN
    C:={CLOSURE({[S′→ · S ， #]})};
    REPEAT
      FOR C 中每个项目集 I 和 G′ 的每个符号 X
      DO
          IF GO(I ， X) 非空且不属于 C ， THEN
                把 GO(I ， X) 加入 C 中
    UNTIL C 不再增大
 END
```

LR(1) 分析表的构造算法

- 构造 LR(1) 分析表的算法
  - 令每个 I_k 的下标 k 为分析表的状态，令含有 [S′→ · S, #] 的 I_k 的 k 为分析器的初态。

LR(1) 分析表的 ACTION 和 GOTO 子表构造

1. 若项目 [A→α · aβ, b] 属于 I_k 且 GO(I_k, a) = I_j，a 为终结符，则置 ACTION[k, a] 为 “ sj” 。
2. 若项目 [A→α · ， a] 属于 I_k，则置 ACTION[k, a] 为 “ rj” ；其中假定 A→α 为文法 G′ 的第 j 个产生式。
3. 若项目 [S′→S · , #] 属于 I_k，则置 ACTION[k, #] 为 “ acc” 。
4. 若 GO(I_k， A) = I_j，则置 GOTO[k, A]=j 。
5. 分析表中凡不能用规则 1 至 4 填入信息的空白栏均填上“出错标志”。

LR(1) 分析表和 LR(1) 文法

- 按上述算法构造的分析表，若不存在多重定义的入口 (即，动作冲突) 的情形，则称它是文法 G 的一张规范的 LR(1) 分析表。
- 使用这种分析表的分析器叫做一个规范的 LR 分析器。
- 具有规范的 LR(1) 分析表的文法称为一个 LR(1) 文法。
- LR(1) 状态比 SLR 多，

$$LR(0) \subset SLR \subset LR(1) \subset \text{无二义文法}$$

示例： LR(1) 分析表构造

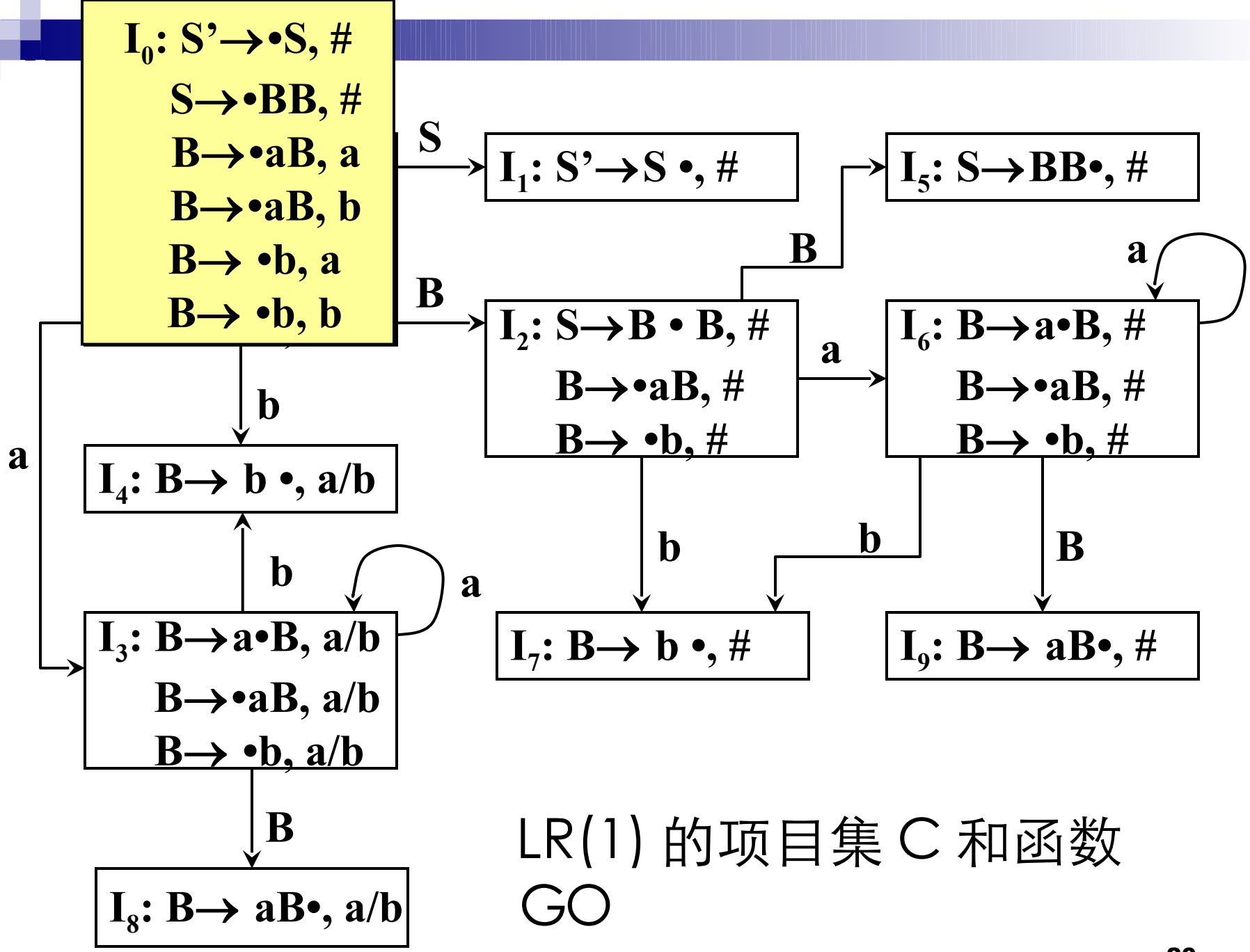
- 例 5.13　(5.10) 的拓广文法 G(S′)

  (0) S′→S

  (1) S→BB

  (2) B→aB

  (3) B→b



LR(1) 的项目集 C 和函数 GO

LR(1) 分析表为：

状态	ACTION			GOTO	
	a	b	#	*S*	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

- 例： 按上表对 aabab 进行分析

步骤	状态	符号	输入串
0	0	#	aabab#
1	03	#*a*	abab#
2	033	#aa	bab#
3	0334	#aab	ab#
4	0338	#aaB	ab#

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

状态	ACTION			GOTO	
	a	b	#	*S*	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7					

例：按上表对 aabab 进行分析

步骤	状态	符号	输入串
4	0338	#aaB	ab#
5	038	#aB	ab#
6	02	#B	ab#
7	026	#Ba	b#
8	0267	#Bab	#

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

状态	ACTION			GOTO	
	a	b	#	*S*	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			

- 例：按上表对 aabab 进行分析

步骤	状态	符号	输入串	
8	0267	#Bab	#	
9	0269	#BaB	#	
10	025	#BB	#	
11	01	#S	#	acc

状态	ACTION			GOTO	
	a	b	#	*S*	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

- 例：按上表对 abab 进行分析

步骤	状态	符号	输入串	
0	0	#	abab#	
1	03	#*a*	bab#	
2	034	#ab	ab#	
3	038	#aB	ab#	
4	02	#B	ab#	
5	026	#Ba	b#	
6	0267	#Bab	#	
7	0269	#BaB	#	
8	025	#BB	#	
9	01	#*S*	#	acc

一些分析器产生器

- YACC 工具实习
  - 阅读资料
    - Yacc 与 Lex 快速入门 .pdf
    - Bison Manual
      - 2.1 Reverse Polish Notation Calculator

一些分析器产生器

- YACC 工具实习
  - 用 Bison 构建一个 Reverse Polish Notation Calculator
    - 编写 rpcalc.y
    - 用 Bison 编译 rpcalc.y，生成 rpcalc.tab.c
    - 用 C++ 编译器编译 rpcalc.tab.c ，生成 rpcalc
    - 运行 rpcalc，输入若干后缀式，观察运行结果
  - 改进
    - 用 Flex 生成词法分析器

一些分析器产生器

- YACC 工具实习
  - 提交文档 .zip
    - rpcalc.y
    - rpcalc.tab.c
    - rpcalc
    - Word 文件 Run.doc: 运行 rpcalc，输入若干后缀式，运行结果的截图
    - rpcalc 的 LEX 源程序： rpcalc.lex
    - rpcalc 词法分析程序 C 源程序： rpcalc.yy.c

一些分析器产生器

- The Lex & Yacc Page
  - http://dinosaur.compilertools.net/
- YACC——Yet Another Compiler Compiler
  - Stephen C. Johnson. YACC: Yet Another Compiler-Compiler. *Unix Programmer's Manual* Vol 2b, 1979.
  - GNU bison: 基本兼容 Yacc，与 flex 一起使用
  - Berkeley Yacc : 目前最好的 Yacc 变种
  - LALR(1) 分析

小结

- LR(1) 项目集规范族
- LR(1) 分析表的构造

作业

- P134—8(选作)

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法