编译原理与技术

--自底向上的语法分析IV

陈俊洁

天津大学智算学部

Outline

- 自底向上的语法分析基本问题
  - 移动－归约分析法
  - 用栈实现移动归约分析
- 算符优先分析法
  - 算符优先分析法定义、优先分析表的确定、优先函数的定义
  - 使用算符优先关系进行分析
  - 算符优先分析中的错误恢复
- LR分析法
  - LR(0)
  - SLR
  - LR(1)
- 语法分析器的自动产生工具Yacc

SLR分析表的构造

LR(0)文法的局限性

- 假定一个LR(0)规范族中含有如下的一个项目集（状态）
  I={X→α•bβ, A→α•, B→α•}
- 第一个项目是移进项目，第二、三个项目是归约项目。这三个项目告诉我们应该做的动作各不相同，互相冲突：
  - 第一个项目告诉我们应该把下一个符号b移进；
  - 第二个项目告诉我们应该把栈顶的α归约为A；
  - 第三个项目告诉我们应该把栈顶的α归约为B。

SLR语法分析概述

- LR(0)文法的活前缀识别自动机的每一个状态（项目集）都不含冲突的项目。
- 对于某些含有冲突的状态，一种带有简单“展望”的LR分析法，即SLR文法，可以解决冲突。
- SLR文法构造分析表的主要思想是：许多冲突性的动作都可能通过考察有关非终结符的FOLLOW集而获解决。

Review:构造结合FOLLOW的算法

对文法G的每个非终结符A构造FOLLOW(A)的办法是：连续应用下列规则,直到每个后随符号集FOLLOW不再增大为止.

1) 对于文法的开始符号S, 置#于FOLLOW(S)中;

2)若A→αBβ是一个产生式, 则把FIRST(β)\{ε}加至FOLLOW(B)中;

3)若A→ αB是一个产生式, 或A→ αBβ是一个产生式而β => ε (即ε∈FIRST(β)), 则把FOLLOW(A)加至FOLLOW(B)中.

SLR基本算法

- 解决冲突的方法是分析所有含A和B的句型，考察集合FOLLOW(A)和FOLLOW(B)，如果这两个集合不相交，而且也不包含b，那么当状态I面临输入符号a时，我们可以使用如下策略：
  - 若a=b，则移进。
  - 若a∈FOLLOW(A)，则用产生式A→α进行归约；
  - 若a∈FOLLOW(B)，则用产生式B→α进行归约；
  - 此外，报错

SLR基本算法

- 假定LR(0)规范族的一个项目集I中
  - 含有m个移进项目
    - $A_1 \rightarrow \alpha \bullet a_1\beta_1$，$A_2 \rightarrow \alpha \bullet a_2\beta_2$，…，$A_m \rightarrow \alpha \bullet a_m\beta_m$；
  - 同时含有n个归约项目
    - $B_1 \rightarrow \alpha \bullet$，$B_2 \rightarrow \alpha \bullet$，…，$B_3 \rightarrow \alpha \bullet$，
  - 如果集合{ a_1,…, a_m}，FOLLOW(B_1)，…，FOLLOW(B_n)两两不相交（包括不得有两个FOLLOW集合有#），则隐含在I中的动作冲突可以通过检查现行输入符号a属于上述n+1个集合中的哪个集合而解决：
    - 若a是某个a_i，i=1,2,…,m，则移进。
    - 若a∈FOLLOW(B_i)，i=1,2,…,m，则用产生式$B_i \rightarrow \alpha$进行归约；
    - 此外，报错

  这种冲突的解决方法叫做SLR(1)解决办法。

SLR语法分析表的构造算法

- 首先把G拓广为G’，对G’构造LR(0)项目集规范族C和活前缀识别自动机的状态转换函数GO。函数ACTION和GOTO可按如下方法构造：
  - 若项目A→α•aβ属于I_k，GO(I_k,a)= I_j, a为终结符，置ACTION[k,a]为“把状态j和符号a移进栈”，简记为“sj”；
  - 若项目A→α•属于I_k，那么，对任何非终结符a，a∈FOLLOW(A)，置ACTION[k,a]为“用产生式A→α进行归约”，简记为“rj”；其中，假定A→α为文法G’的第j个产生式
  - 若项目S’→S•属于I_k，则置ACTION[k,#]为可“接受”，简记为“acc”；
  - 若GO(I_k, A)= I_j，A为非终结符，则置GOTO[k, A]=j；
  - 分析表中凡不能用规则1至4填入信息的空白格均填上“出错标志”。
  - 语法分析器的初始状态是包含S’→•S的项目集合的状态

按上述方法构造的含有Action和GOTO的分析表，若每个入口不含多重定义，则称它为文法G的一张SLR表。具有SLR表的文法G成为一个SLR(1)文法，使用SLR表的分析器叫SLR分析器。

SLR分析举例

每个SLR(1)文法都是无二义的；
但并非每个无二义的文法都是SLR(1)的。

文法G：

(0) S’ → E
(1) E → E+T
(2) E → T
(3) T → T*F
(4) T → F
(5) F → (E)
(6) F → i

SLR分析举例

每个SLR(1)文法都是无二义的；
但并非每个无二义的文法都是SLR(1)的。

文法G：

(0) S’ → E
(1) E → E+T
(2) E → T
(3) T → T*F
(4) T → F
(5) F → (E)
(6) F → i

I_0:S’ → •E
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •i

I_1:S’ → E•
E → E•+T

I_2:E → T•
T → T•*F

I_3:T → F•

I_4:F → (•E)
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •i

I_5:F → i•

I_6:E → E+•T
T → •T*F
T → •F
F → •(E)
F → •i

I_7:T → T*•F
F → •(E)
F → •i

I_8:F → (E•)
E → E•+T

I_9:E → E+T•
T → T•*F

I_{10}:T → T*F•

I_{11}:F → (E)•

FOLLOW(S’)={#}
FOLLOW(E)={#,), +}

SLR分析举例

文法G：

(0) S’ → E　　(1) E → E+T

(2) E → T　　(3) T → T*F

(4) T → F　　(5) F → (E)

(6) F → i

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	S5			S4			1	2	3
1		S6				acc			
2		r2	S7		r2	r2			
3		r4	r4		r4	r4			
4	S5			S4			8	2	3
5		r6	r6		r6	r6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		r1	S7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

I_1:S’ → E•
E → E•+T

I_2:E → T•
T → T•*F

I_9:E → E+T•
T → T•*F

FOLLOW(S’)={#}
FOLLOW(E)={#,), +}

I_0:S’ → •E
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •i

I_1:S’ → E•
E → E•+T

I_2:E → T•
T → T•*F

I_3:T → F•

I_4:F → (•E)
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •i

I_5:F → i•

I_6:E → E+•T
T → •T*F
T → •F
F → •(E)
F → •i

I_7:T → T*•F
F → •(E)
F → •i

I_8:F → (E•)
E → E•+T

I_9:E → E+T•
T → T•*F

I_{10}:T → T*F•

I_{11}:F → (E)•

构造规范LR语法分析表

SLR语法分析的局限性

- 所有的SLR语法必须满足如下条件
  - I = {X →α • bβ , A →α • , B →α • }
    若有：FOLLOW(A) ∩FOLLOW(B) = Ø
    FOLLOW(A) ∩{b} = Ø
    FOLLOW(B) ∩{b} = Ø
- 状态I面临某输入符号a
  1) 若a=b，则移进
  2) 若a∈FOLLOW(A), 则用产生式 A →γ 进行归约
  3) 若a∈FOLLOW(B), 则用产生式 B →δ 进行归约
  4) 此外，报错

SLR文法中可能出现的冲突

文法G'：

(0) S' → S
(1) S → L=R
(2) S → R
(3) L → *R
(4) L → i
(5) R → L

I_0:S' → •S
S → •L=R
S → •R
L → •*R
L → •i
R → •L

I_1:S' → S•

I_2:S → L•=R
R → L•

I_3:S → R•

I_4:L → *•R
R → •L
L → •*R
L → •i

I_5:L → i•

I_6:S → L=•R
R → •L
L → •*R
L → •i

I_7:L → *R•

I_8:R → L•

I_9:S → L=R•

= ∈ FOLLOW(R)

构造规范LR语法分析表

- 针对SLR语法分析的局限性，我们给出如下解决方案：
  - 重新定义项目，使之包含一个终结符串作为第二个分量，可以把更多的信息并入状态中。
  - 项目的一般形式也就变成了[A→α•β, $a_1a_2\cdots a_k$]，其中A→α•β 是LR(0)项目，每一个a都是终结符或者#。——LR(k)
  - 项目中的$a_1a_2\cdots a_k$称为它的向前搜索符串（或展望串）
  - 这样的a的集合是FOLLOW(A)的子集，有可能是真子集
  - 归约项目[A→α•, $a_1a_2\cdots a_k$]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，才可以把栈顶的α归约为A。我们只对k⩽1的情形感兴趣 ——LR(1)

LR(1)对活前缀有效的定义

- LR(1)的项目[A→α·β, a]对活前缀γ有效, 如果存在规范推导$S \overset{*}{=>} \delta A\omega => \delta\alpha\beta\omega$：
  - γ=δα
  - a是ω的第一个符号，或者ω是空串，a是#。
- Closure运算的新定义
  - 考虑对活前缀γ有效的项目集中的项目[A→α·Bβ, a]必定存在一个最右推导$S \overset{*}{=>} \delta Aax => \delta\alpha B\beta ax$，其中γ=δα
  - 假设βax能推导出终结符串bw, 那么对每一个形如B→η的产生式, 存在推导$S \overset{*}{=>} \gamma Bbw => \gamma \eta bw$,于是[B→·η, b]对γ有效, 其中b是FIRST(βax)中的任何终结符。根据FIRST的定义，FIRST(βax)= FIRST(βa)

假定I是文法G'的任一项目集，构造I的闭包CLOSURE(I)的方法是：

- I的任何项目都属于CLOSURE(I)；
- 若[A→α•Bβ, a]属于CLOSURE(I)，B→η是一个产生式，那么，对FIRST(βa)中的每个终结符b, 将[B→•η, b]加入CLOSURE(I)；
- 重复上述两步骤直至CLOSURE(I)不再增大为止。

函数值GO(I,X)定义为GO(I,X)=CLOSURE(J)，其中J={任何形如[A→αX•β, a]的项目 | [A→α•Xβ, a]属于I}

LR(1)分析表构造举例

文法G：

(0) S’ → S

(1) S → CC

(2) C → cC

(3) C → d

LR(1)分析表构造举例

文法G：

(0) S' → S
(1) S → CC
(2) C → cC
(3) C → d

I_0: S' → •S, #
S → •CC, #
C → •cC, c/d
C → •d, c/d

I_1: S' → S•, #

I_2: S → C•C, #
C → •cC, #
C → •d, #

I_3: C → c•C, c/d
C → •cC, c/d
C → •d, c/d

I_4: C → d•, c/d

I_5: S → CC•, #

I_6: C → c•C, #
C → •cC, #
C → •d, #

I_7: C → d•, #

I_8: C → cC•, c/d

I_9: C → cC•, #

GO(I_0, S)=I_1;
GO(I_0, C)=I_2;
GO(I_0, c)=I_3;
GO(I_0, d)=I_4;
GO(I_2, C)=I_5;
GO(I_2, c)=I_6;
GO(I_2, d)=I_7;

GO(I_3, C)=I_8;
GO(I_3, c)=I_3;
GO(I_3, d)=I_4;
GO(I_6, C)=I_9;
GO(I_6, c)=I_6;
GO(I_6, d)=I_7;

规范LR语法分析表的构造

- 步骤
  - 构造拓广文法G’ 的LR(1)项目集规范族C={I_0,I_1,…,I_n}
  - 从I_k构造语法分析器的状态k，状态k的分析动作如下：
    - 如果[A→α·aβ, b]在I_k中，且GO(I_k, a)= I_j，则置action[k, a]为s_j，即“移动(j, a)进栈”，这里要求a必须是终结符
    - 如果[A→α·, a]在I_k中，则置action[k, a]为r_j，即按照r_j归约，其中j是产生式A→α的序号
    - 如果[S’ →S·, #]在I_k中，则置action[k, #]为acc，表示接受
    - 状态k的转移按照下面的方法确定：如果GO(I_k, A)= I_j,那么goto[k, A]=j
    - 其余表项设为出错
  - 初始状态是包含[S’ →·S, #]的项目集构造出的状态。

如果该表无冲突，那么称该表为G的规范LR(1)分析表。具有规范的LR(1)分析表的文法称为一个LR(1)文法。

LR(1)分析表构造举例

状态	ACTION			GOTO	
	c	d	#	S	C
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		

I_0:S' → •S, #
S → •CC, #
C → •cC, c/d
C → •d, c/d

I_1:S' → S•, #

I_2:S → C•C, #
C → •cC, #
C → •d, #

I_3:C → c•C, c/d
C → •cC, c/d
C → •d, c/d

I_4:C → d•, c/d

I_5:S → CC•, #

I_6:C → c•C, #
C → •cC, #
C → •d, #

I_7:C → d•, #

I_8:C → cC•, c/d

I_9:C → cC•, #

文法G：
(0) S' → S
(1) S → CC
(2) C → cC
(3) C → d

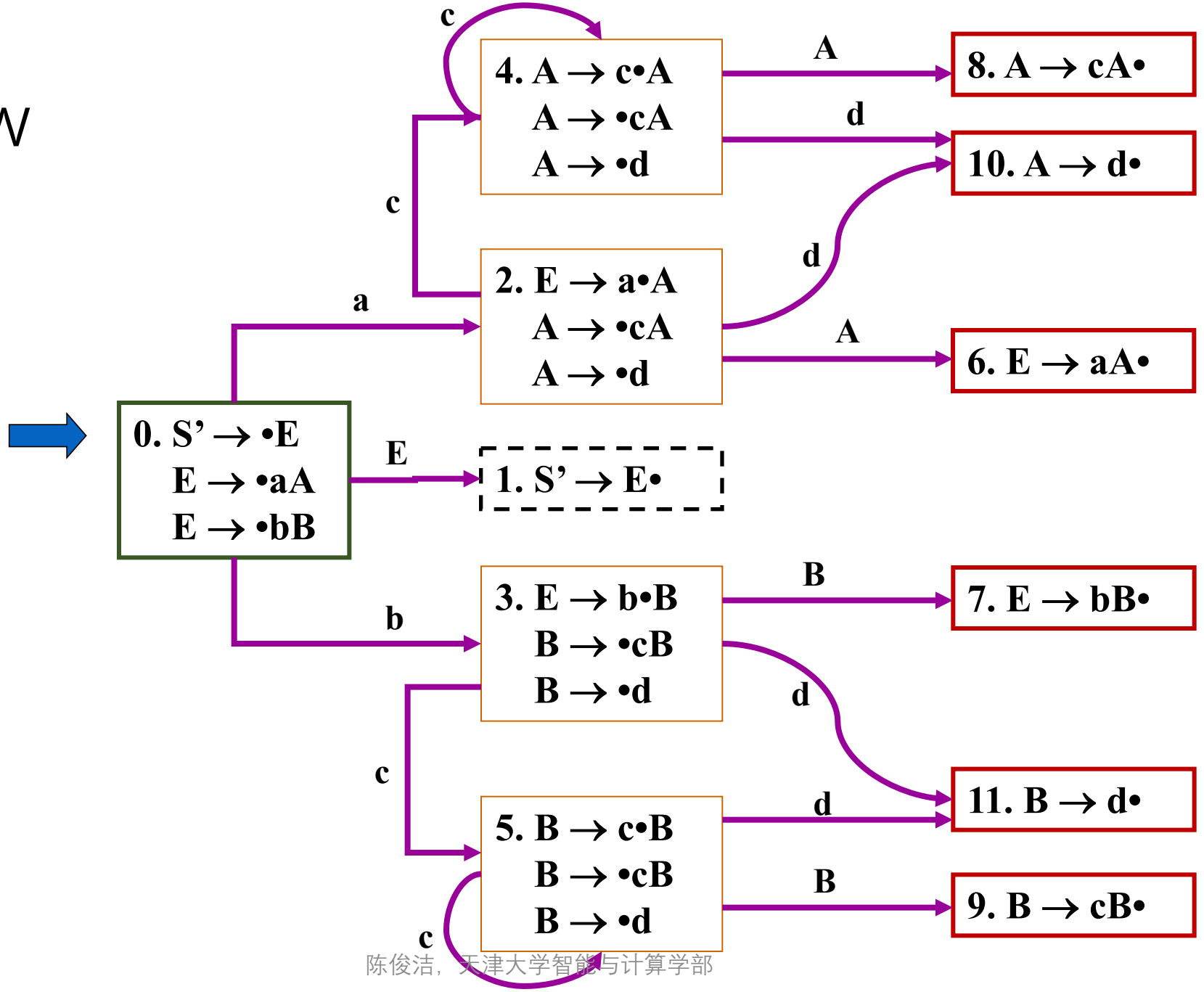
$GO(I_0, S)=I_1$;
$GO(I_0, C)=I_2$;
$GO(I_0, c)=I_3$;
$GO(I_0, d)=I_4$;
$GO(I_2, C)=I_5$;
$GO(I_2, c)=I_6$;
$GO(I_2, d)=I_7$;

$GO(I_3, C)=I_8$;
$GO(I_3, c)=I_3$;
$GO(I_3, d)=I_4$;
$GO(I_6, C)=I_9$;
$GO(I_6, c)=I_6$;
$GO(I_6, d)=I_7$;

Review

文法G：

(0) S’ → E
(1) E → aA
(2) E → bB
(3) A → cA
(4) A → d
(5) B → cB
(6) B → d



LR(0)分析表

状态	action					goto		
	a	*b*	*c*	*d*	#	*E*	*A*	*B*
0	*s2*	s3				*1*		
1					*acc*			
2			*s4*	*s10*			*6*	
3			*s5*	*s11*				*7*
4			*s4*	*s10*			*8*	
5			*s5*	*s11*				*9*
6	*r1*	*r1*	*r1*	*r1*	*r1*			
7	*r2*	*r2*	*r2*	*r2*	*r2*			
8	*r3*	*r3*	*r3*	*r3*	*r3*			
9	*r5*	*r5*	*r5*	*r5*	*r5*			
10	*r4*	*r4*	*r4*	*r4*	*r4*			
11	*r6*	*r6*	*r6*	*r6*	*r6*			

LR(0)文法的局限性

- 假定一个LR(0)规范族中含有如下的一个项目集（状态）
  I={X→α•bβ, A→α•, B→α•}
- 第一个项目是移进项目，第二、三个项目是归约项目。这三个项目告诉我们应该做的动作各不相同，互相冲突：
  - 第一个项目告诉我们应该把下一个符号b移进；
  - 第二个项目告诉我们应该把栈顶的α归约为A；
  - 第三个项目告诉我们应该把栈顶的α归约为B。

SLR分析举例

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

I_1:S' → E•
E → E•+T

I_2:E → T•
T → T•*F

I_9:E → E+T•
T → T•*F

文法G：
(0) S' → E
(1) E → E+T
(2) E → T
(3) T → T*F
(4) T → F
(5) F → (E)
(6) F → i

FOLLOW(S')={#}
FOLLOW(E)={#,), +}

SLR语法分析的局限性

- 所有的SLR语法必须满足如下条件
  - $I = \{X \rightarrow \alpha \bullet b\beta\ ,\ A \rightarrow \alpha \bullet\ ,\ B \rightarrow \alpha \bullet\}$
    若有：$FOLLOW(A) \cap FOLLOW(B) = \varnothing$
    $FOLLOW(A) \cap \{b\} = \varnothing$
    $FOLLOW(B) \cap \{b\} = \varnothing$

文法G'：

(0) S' → S
(1) S → L=R
(2) S → R
(3) L → *R
(4) L → i
(5) R → L

I_2:S → L•=R
R → L•

= ∈ FOLLOW(R)

LR(1)对活前缀有效的定义

- LR(1)的项目[A→α·β, a]对活前缀γ有效, 如果存在规范推导$S \overset{*}{=>} \delta A\omega => \delta\alpha\beta\omega$：
  - γ=δα
  - a是ω的第一个符号，或者ω是空串，a是#。
- Closure运算的新定义
  - 考虑对活前缀γ有效的项目集中的项目[A→α·Bβ, a]必定存在一个最右推导$S \overset{*}{=>} \delta Aax => \delta\alpha B\beta ax$，其中γ=δα
  - 假设βax能推导出终结符串bw, 那么对每一个形如B→η的产生式, 存在推导$S \overset{*}{=>} \gamma Bbw => \gamma \eta bw$,于是[B→·η, b]对γ有效, 其中b是FIRST(βax)中的任何终结符。根据FIRST的定义，FIRST(βax)= FIRST(βa)

假定I是文法G'的任一项目集，构造I的闭包CLOSURE(I)的方法是：

- I的任何项目都属于CLOSURE(I)；
- 若[A→α•Bβ, a]属于CLOSURE(I)，B→η是一个产生式，那么，对FIRST(βa)中的每个终结符b, 将[B→•η, b]加入CLOSURE(I)；
- 重复上述两步骤直至CLOSURE(I)不再增大为止。

函数值GO(I,X)定义为GO(I,X)=CLOSURE(J)，其中J={任何形如[A→αX•β, a]的项目 | [A→α•Xβ, a]属于I}

LR(1)分析表构造举例

状态	ACTION			GOTO	
	c	d	#	S	C
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		

I_0:S' → •S, #
S → •CC, #
C → •cC, c/d
C → •d, c/d

I_1:S' → S•, #

I_2:S → C•C, #
C → •cC, #
C → •d, #

I_3:C → c•C, c/d
C → •cC, c/d
C → •d, c/d

I_4:C → d•, c/d

I_5:S → CC•, #

I_6:C → c•C, #
C → •cC, #
C → •d, #

I_7:C → d•, #

I_8:C → cC•, c/d

I_9:C → cC•, #

文法G：
(0) S' → S
(1) S → CC
(2) C → cC
(3) C → d

$GO(I_0, S)=I_1$;
$GO(I_0, C)=I_2$;
$GO(I_0, c)=I_3$;
$GO(I_0, d)=I_4$;
$GO(I_2, C)=I_5$;
$GO(I_2, c)=I_6$;
$GO(I_2, d)=I_7$;

$GO(I_3, C)=I_8$;
$GO(I_3, c)=I_3$;
$GO(I_3, d)=I_4$;
$GO(I_6, C)=I_9$;
$GO(I_6, c)=I_6$;
$GO(I_6, d)=I_7$;

	项目	项目集规范族	Action表 (遇规约项目)
LR(0)	LR(0)		整行填rj
SLR	LR(0)		看Follow集合
LR(1)	LR(1)	Closure 第二个分量算First(βa)	看项目的第二个分量

二义性文法在LR分析中的应用

- 定理：任何二义性文法不是LR文法，因而也不是SLR或LALR文法
- 算术表达式的二义性文法
  - E⟶E+E | E*E | (E) | id
- 对应的非二义性文法为
  - E⟶E+T | T
  - T⟶T*F | F
  - F⟶(E) | id

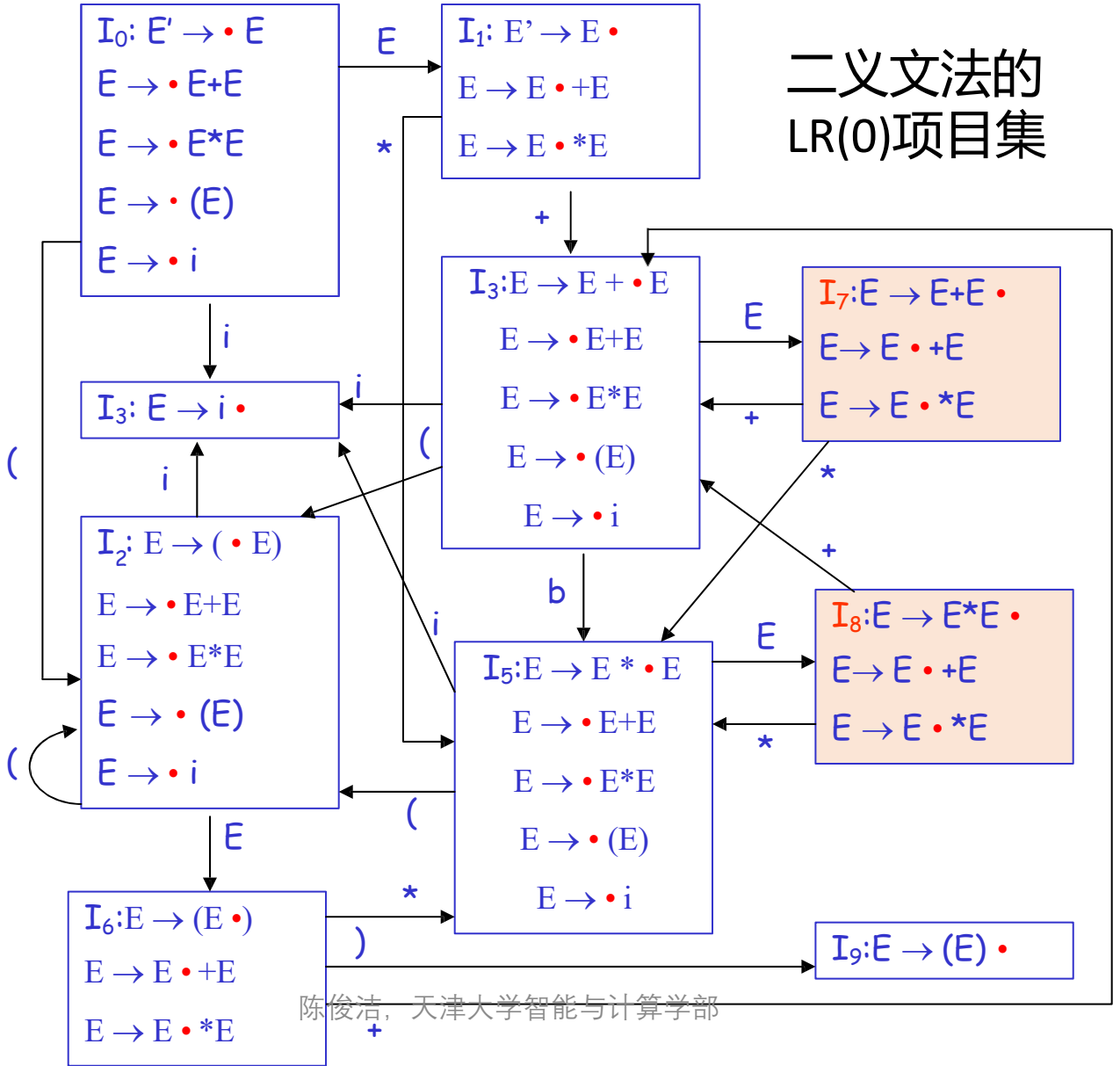
I_7: E → E+E•
E → E•+E
E → E•*E

I_8: E → E*E•
E → E•+E
E → E•*E

对于某些二义文法，可以人为地给出优先性和结合性的规定，从而可以构造出比相应非二义性文法更优越的LR分析器

二义性文法在LR分析中的应用

文法：
(1) E → E+E
(2) E →E*E
(3) E → (E)
(4) E → i



二义文法的
LR(0)项目集

二义性文法在LR分析中的应用

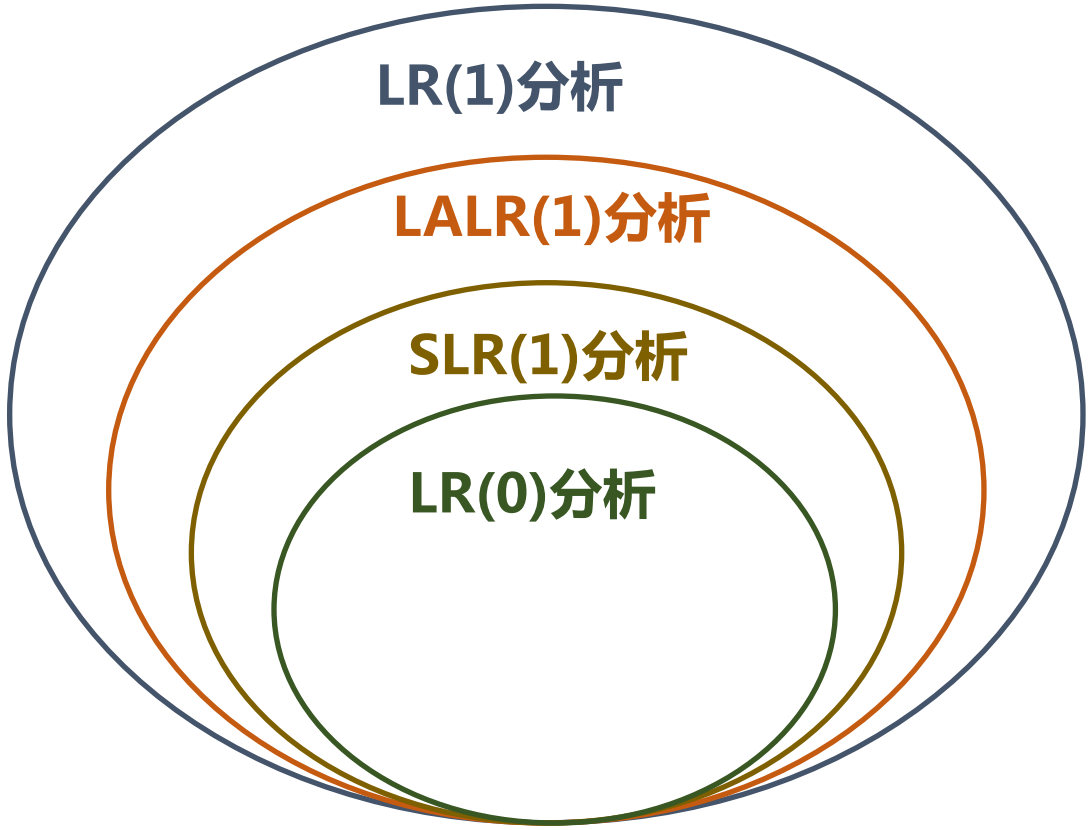
- 定义*优先于+; *、 + 服从左结合，得到二义文 法的LR分析表

状态	ACTION						GOTO
	i	+	*	(	)	#	E
0	S_3			S_2			1
1		S_4	S_5			Acc	
2	S_3			S_2			6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			7
5	S_3			S_2			8
6		S_4	S_5		S_9		
7		r_1	S_5		r_1	r_1	
8		r_2	r_2		r_2	r_2	
9		r_3	r_3		r_3	r_3	

LR语法分析中的错误恢复

- 在LR分析过程中，当我们处在这样一种状态下，即输入符号既不能移入栈顶，栈内元素又不能归约时，就意味着发现语法错误。
- 处理的方法分为两类：
  - 第一类多半是用插入、删除或修改的办法。如果不能使用这种办法，则采用第二类办法，
  - 第二类办法包括在检查到某一不合适的短语时，采用局部化的方法进行处理。类似前面讲过的同步符号

LR分析

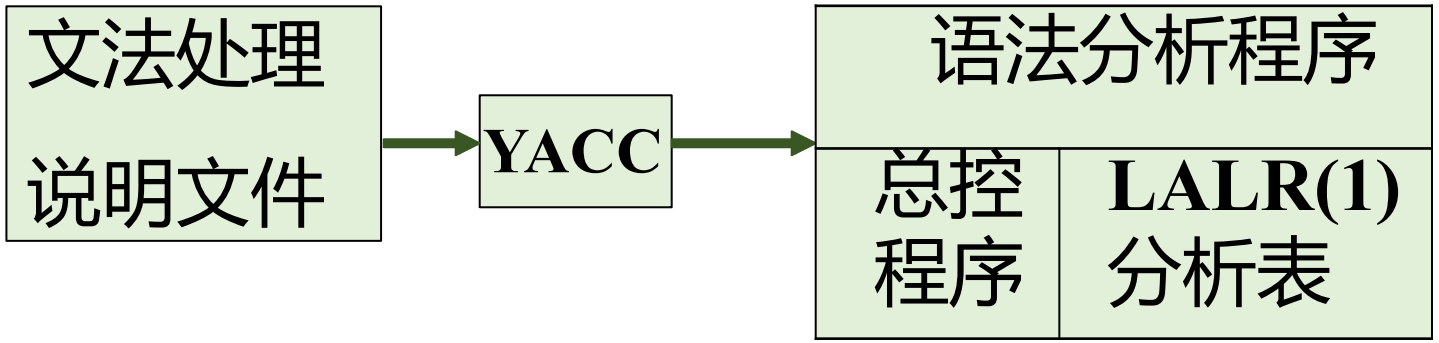


Outline

- 自底向上的语法分析基本问题
  - 移动－归约分析法
  - 用栈实现移动归约分析
- 算符优先分析法
  - 算符优先分析法定义、优先分析表的确定、优先函数的定义
  - 使用算符优先关系进行分析
  - 算符优先分析中的错误恢复
- LR分析法
- 语法分析器的自动产生工具Yacc

语法分析器自动产生工具Yacc

- 软件工具Yacc是编译程序自动生成器，是在UNIX中运行的一个实用程序.该程序读用户提供的关于语法分析器的规格说明,基于LALR(1)语法分析的原理,自动构造一个 语法分析器;并且能根据规格说明中给出的 语义动作建立规定的翻译.



Yacc语言程序的组成

- **Yacc**语言程序,与**LEX**规格说明类似, 由说明部分、 翻译规则和辅助过程组成.
- 各部分之间用双百分号分隔即:

说明部分---可有可无 (包括变量说明,标识符常量说明和正规定义.)

%%

翻译规则 (必须有)

%%

P1 {动作1}
P2 {动作2}
……
Pn {动作n}

辅助过程---可有可无 (对翻译规则的补充, 翻译规则里某些动作需要调用过程, 如C语言的库程序.)

Yacc例子

- 其中**digit**表示**0…9**的数字,按如下文法写出的**Yacc**规格说明如下**:**

E→ E+T|T

T→ T*F|F

F→ (E)|digit

```
(1)   %{
(2)      # include      <ctype.h>
(3)      %}
(4)      token DIGIT  (5)
      %%
```

第(1)-(4)行是说明部分,其中包括可供 选择的两部分.

用%{和%}括起来的部分是C 语言程序的正规说明,可以说明翻译规则和 辅助过程里使用的变量和函数的类型.

这里用第(2)行的蕴含控制行代替全部说明,具体内容在 ctype.h里,第(4)行指出DIGIT是token类型的词汇,供后面使用.

Yacc例子

```
(6)   line    :  expr '\n'    { printf("%d\n", $1); }
(7)           ;
(8)   expr    :  expr '+' term      { $$ = $1 + $3; }
(9)           |  term
(10)          ;
(11)  term    :  term     '*'       { $$ = $1 * $3; }
(12)             factor
(13)          |  factor
(14) factor   ;  :  '('  ' expr ')'    { $$ = $2 ; }
(15)          |  DIGIT
(16)          ;
(17)%%
```

第(6)-(16)行是翻译规则,每条规则由 文法的产生式和相关的语义动作组成.
形如: 左部→右部1|右部2|…|右部n 的产生式,在YACC规格说明里写成

```
左部:    右部1   {语义动作1}
     |    右部2   {语义动作2}
     ……
     |    右部n   {语义动作n}
     ;
```

在YACC里,用单引号括起来的单个字符看成是终结符号。语义动作是C语言的语句序列。语义动作中,$$表示和左部非终结符相关的属性值, $1表示和产生式第一个文 法符号相关的属性值,例如: 语义动作
line: expr '\n' { printf("%d\n",$1); }

Yacc例子

```
(18) yylex( ) {
(19)            int     c ;
(20)            c = getchar ( );
(21)            if   (   isdigit ( c ))          {
(22)                 yylval=c-'0';
(23)                 return DIGIT;
(24)            }
(25)            return       c;
(26)       }
```

(18)-(26)行是辅助过程,每个辅助过程都是C语言的函数,对翻译规则的补充,并且,其中必须包含名为yylex的词法分析器。调用名为yylex( )的函数得到一个词汇，该词汇包括两部分的属性，其中值通过Yacc定义的全程变量yylval传递给语法分析器,返回词汇的属性。

(22) yylval=c-'0'; yylex( )返回词汇的值.

(23) yylex( )返回种别DIGIT;

第(25)行,除了数之外的任何字符, yylex( )返回该字符本身.

Yacc处理冲突的规则

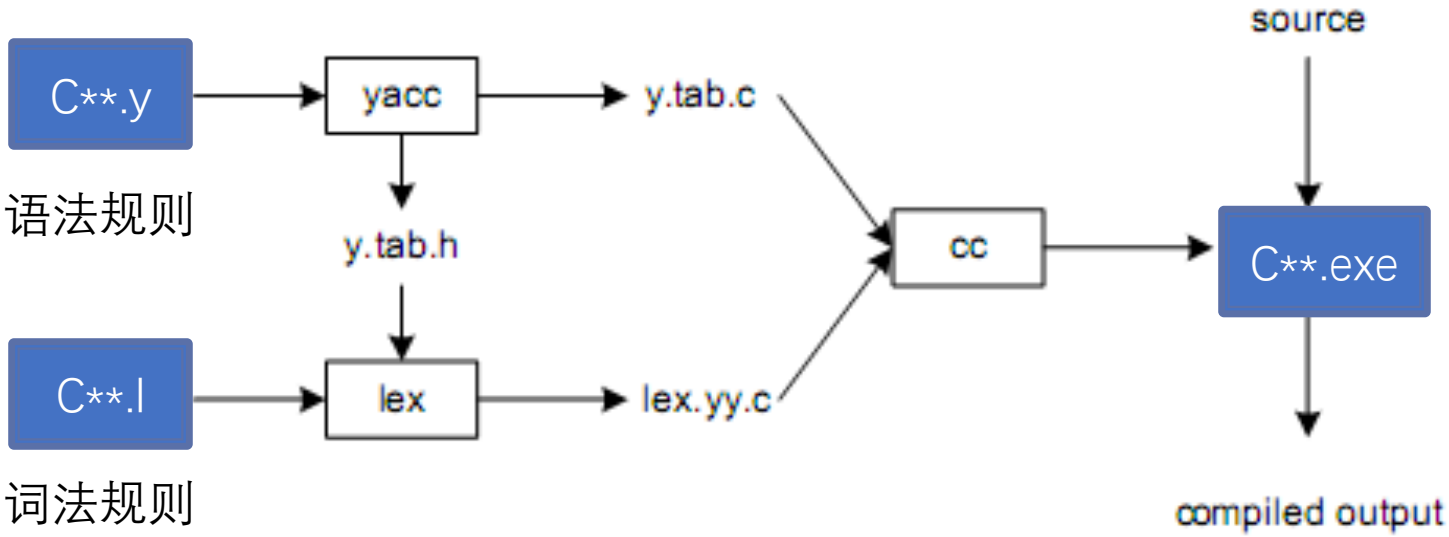
1)产生“归约-归约”.冲突时,按照规格说明中产生式的排列顺序,选择排列在前面的产生式进行归约.

2)当产生“移进-归约”冲突时,选择执行移进动作.

3) Yacc在规格说明部分里,可以规定终结符号的优先顺序和结合性.

- 优先顺序: 按说明终结符的次序,后说明的具有最高的优先顺序. “%prec”说明其后的终结符具有最高的优先顺序.
- 结合性: “%right”(右结合),
  “%left” (左结合),
  “%nonassoc”(不具有结合性)

4) 应用这些机制,对二义文法,用户可以提供 附加信息Yacc可以解决冲突.


C**.y
语法规则
yacc
y.tab.c
y.tab.h
C**.l
词法规则
lex
lex.yy.c
cc
source
C**.exe
compiled output