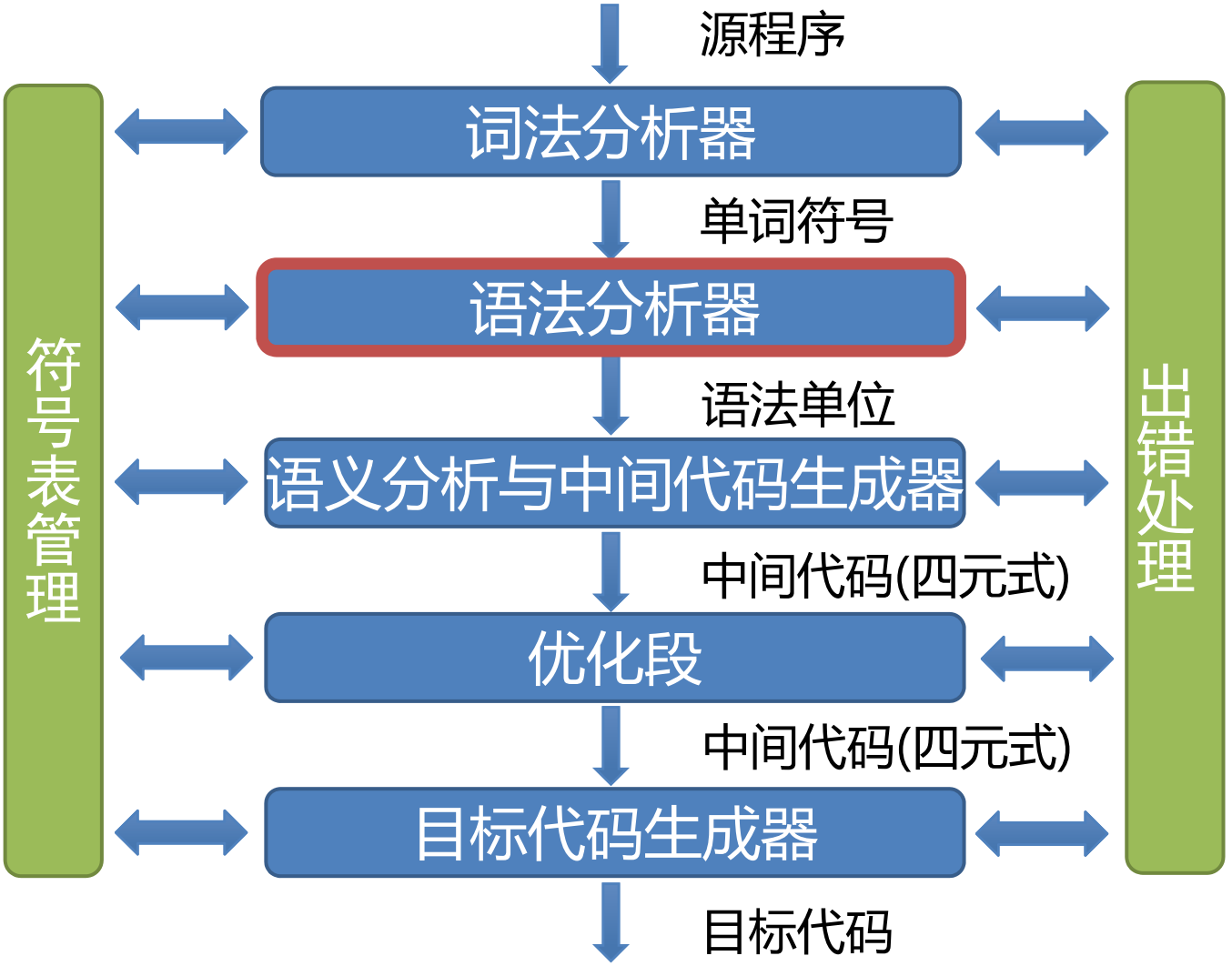
编译原理

更强的LR分析

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

编译原理

一个非LR(0)文法

一个非LR(0)文法

- 拓广文法G(S′):

(0) S′→E

(1) E→E+T

(2) E→T

(3) T→T*F

(4) T→F

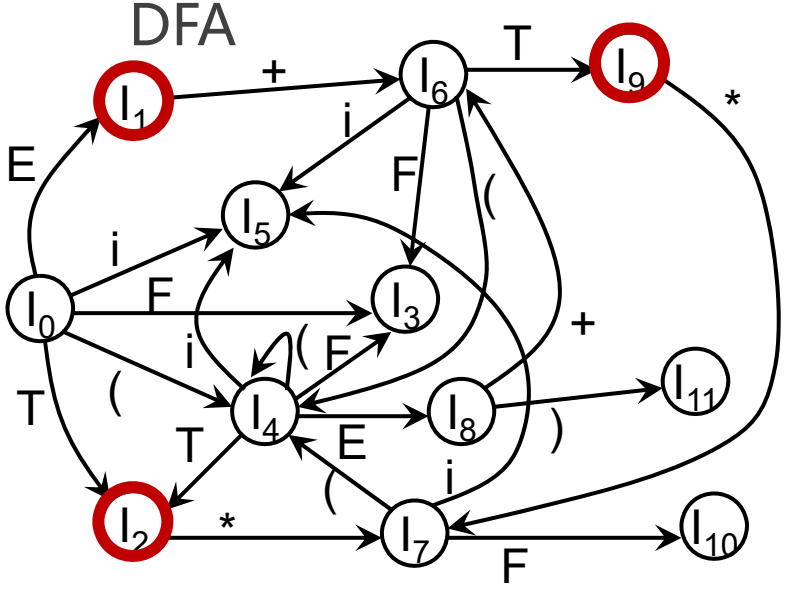
(5) F→(E)

(6) F→i

一个非LR(0)文法

文法G(S′):
(0) S′→E
(1) E→E+T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→(E)
(6) F→i

识别活前缀的DFA



LR(0)项目集规范族

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)•

一个非LR(0)文法

- I_1、I_2和I_9都含有“移进 - 归约”冲突

I_1: S′→E• E→E•+T	I_2: E→T• T→T•*F	I_9: E→E+T• T→T•*F

$$FOLLOW(A) = \left\{ a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T \right\}$$

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

编译原理

SLR(1)冲突解决办法

冲突解决办法

- 假定一个LR(0)规范族含有如下的一个项目集(状态)I = {X→α•bβ，A→α•，B→α•}
- FOLLOW(A)和FOLLOW(B)的交集为∅，且不包含b
- 当状态I面临任何输入符号a 时，可以:
  1. 若a=b，则移进；
  2. 若a∈FOLLOW(A)，用产生式A→α进行归约；
  3. 若a∈FOLLOW(B)，用产生式B→α进行归约；
  4. 此外，报错。

SLR(1)冲突解决办法

- 假定LR(0)规范族的一个项目集I={$A_1 \rightarrow \alpha \bullet a_1\beta_1$，$A_2 \rightarrow \alpha \bullet a_2\beta_2$，…，$A_m \rightarrow \alpha \bullet a_m\beta_m$，$B_1 \rightarrow \alpha \bullet$，$B_2 \rightarrow \alpha \bullet$，…，$B_n \rightarrow \alpha \bullet$ } 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#)，则当状态I面临任何输入符号a 时：
  1. 若a是某个a_i，i=1,2,…,m，则移进；
  2. 若a∈FOLLOW(B_i)，i=1,2,…,n，则用产生式$B_i \rightarrow \alpha$进行归约；
  3. 此外，报错。
- SLR(1)解决办法： S-Simple， 1-最多向前看一个单词

编译原理

SLR(1)分析表的构造

构造SLR(1)分析表的方法

- 把G拓广为G′
- 对G′构造
  - LR(0)项目集规范族C
  - 活前缀识别自动机的状态转换函数GO
- 使用C和GO，构造SLR分析表
  - 令每个项目集I_k的下标k作为分析器的状态，包含项目S′→•S的集合Ik的下标k为分析器的初态。
  - 构造分析表的ACTION和GOTO子表

SLR(1)分析表的ACTION和GOTO子表构造

1. 若项目$A\rightarrow\alpha\bullet a\beta$属于$I_k$且$GO(I_k,a)=I_j$，a为终结符，则置ACTION[k,a] “sj”；
2. 若项目$A\rightarrow\alpha\bullet$属于$I_k$，那么，对任何终结符$a\in FOLLOW(A)$，置ACTION[k,a]为 “rj”；其中，假定$A\rightarrow\alpha$为文法G′的第j个产生式；
3. 若项目$S'\rightarrow S\bullet$属于I_k，则置ACTION[k,#]为“acc”；
4. 若$GO(I_k,A)=I_j$，A为非终结符，则置GOTO[k,A]=j；
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。

SLR(1)和LR(0)分析表构造方法的对比

SLR(1)分析表的ACTION和GOTO子表构造

1. 若项目A→α•aβ属于I_k且GO(I_k,a)=I_j，a为终结符，则置ACTION[k,a] “sj” ；
2. 若项目A→α•属于I_k，那么，对任何终结符 a∈FOLLOW(A)，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，A为非终结符，则置GOTO[k,A]=j；
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志” 。

LR(0)分析表的ACTION和GOTO子表构造

1. 若项目A→α•aβ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•属于I_k，那么，对任何终结符a(或结束符#)，置ACTION[k, a]为 “rj”(假定产生式A→α是文法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，A为非终结符，则置GOTO[k, A]=j。
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。

SLR(1)文法

- 按上述方法构造出的ACTION与GOTO表如果不含多重入口，则称该文法为SLR(1)文法。
- 使用SLR表的分析器叫做一个SLR分析器。
- 每个SLR(1)文法都是无二义的。但也存在许多无二义文法不是SLR(1)的。
- LR(0) ⊂ SLR(1) ⊂ 无二义文法

编译原理

SLR(1)分析表构造示例

示例：SLR(1)分析表的构造

- 拓广文法G(S′)：

(0) S′→E

(1) E→E+T

(2) E→T

(3) T→T*F

(4) T→F

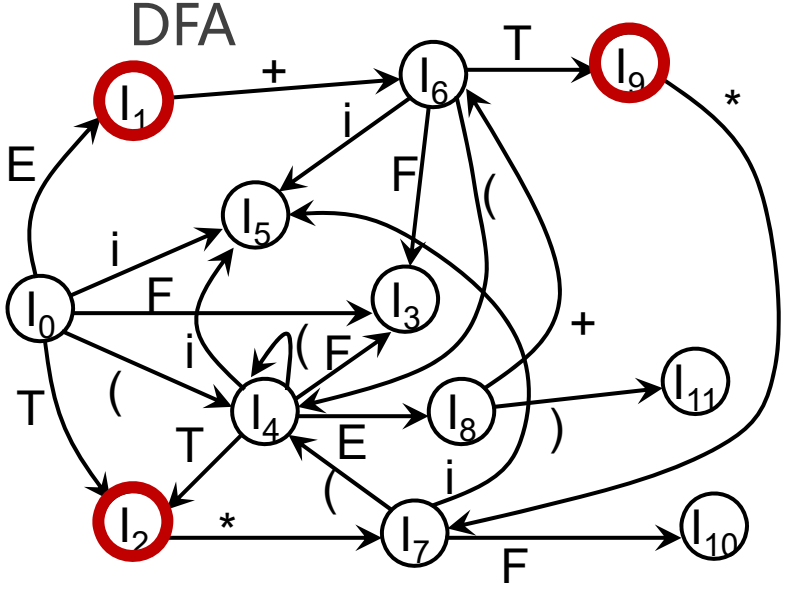
(5) F→(E)

(6) F→i

一个非LR(0)文法

文法G(S′):
(0) S′→E
(1) E→E+T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→(E)
(6) F→i

识别活前缀的DFA



LR(0)项目集规范族

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)•

一个非LR(0)文法

- I_1、I_2和I_9都含有“移进 - 归约”冲突

I_1: S′→E• E→E•+T	I_2: E→T• T→T•*F	I_9: E→E+T• T→T•*F

$$FOLLOW(A) = \left\{ a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T \right\}$$

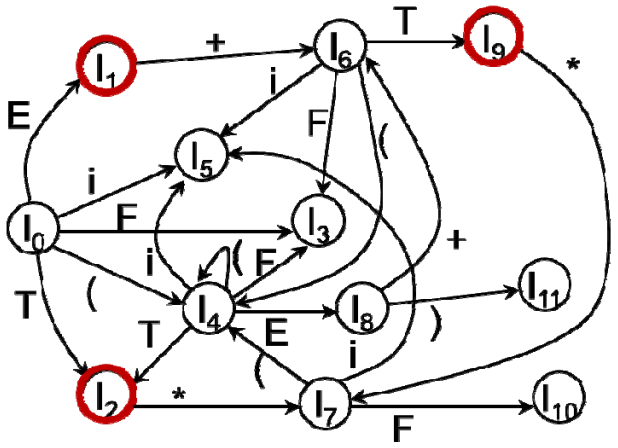
FOLLOW(S′)＝{ # }
FOLLOW(E) ＝{ +,), # }

采取SLR(1)冲突消解

FOLLOW(S′)={ # }
FOLLOW(E)={ +,), # }
FOLLOW(T)={ +, *,), # }
FOLLOW(F)={ +, *,), # }

(0) S′→E
(1) E→E+T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→(E)
(6) F→i

	ACTION						GOTO		
状态	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_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)•

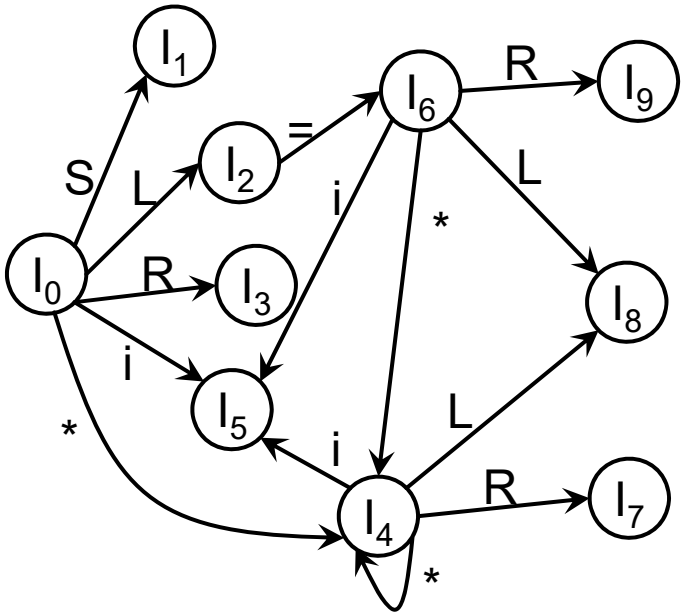
编译原理

一个非SLR(1)文法

SLR冲突消解存在的问题

- 拓广文法G(S′)

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



识别活前缀的DFA

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)={#, =}

LR(0)项目集规范族

没有以“R=”为前缀的规范句型
有以“*R=”为前缀的规范句型

$$FOLLOW(A) = \left\{ a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T \right\}$$

状态栈	符号栈	输入串
02	#L	=…#
03	#R	=…#

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$A\rightarrow\alpha\bullet$ 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，当前单词是a，栈里的活前缀$\beta\alpha$未必允许把α归约为A，因为可能根本就不存在一个形如“βAa”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

$$FOLLOW(A)=\left\{a\middle|S\overset{*}{\Rightarrow}\ldots Aa\ldots,a\in V_T\right\}$$

编译原理

LR(1)分析表的构造

构造LR(1)分析表的方法

- 把G拓广为G′
- 对G′构造LR(1)项目集规范族C和活前缀识别自动机的状态转换函数GO
- 使用C和GO，构造LR(1)分析表

LR(k)项目

- LR(k)项目：扩展LR(0)项目，附带有k个终结符
  $$[A\rightarrow\alpha\bullet\beta,\ a_1a_2...a_k]$$
  $a_1a_2...a_k$ 称为向前搜索符串(或展望串)。
- 归约项目$[A\rightarrow\alpha\bullet,\ a_1a_2...a_k]$的意义：当它所属的状态呈现在栈顶且后续的k个输入符号为 $a_1a_2...a_k$ 时，才可以把栈顶上的α归约为A
- 对于任何移进或待约项目$[A\rightarrow\alpha\bullet\beta,\ a_1a_2...a_k]$，$\beta\neq\varepsilon$，搜索符串 $a_1a_2...a_k$ 没有直接作用

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

$$S' \overset{*}{\Rightarrow}_R \delta A\omega \Rightarrow_R \delta\alpha\beta\omega$$

其中，1) γ = δα；2) a是ω的 第一个符号，或者a为#而ω为ε。

有效项目的性质

- [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

$$S \overset{*}{\Rightarrow}_R \delta A a \Rightarrow_R \delta\alpha B\beta a\omega$$

∵ $b \in FIRST(\beta a)$　　∴ $\beta a\omega \overset{*}{\Rightarrow}_R b\varphi$

若B→ξ是产生式，则

$$S \overset{*}{\Rightarrow}_R \delta\alpha B\beta a\omega \overset{*}{\Rightarrow}_R \delta\alpha Bb\varphi \Rightarrow_R \delta\alpha\xi b\varphi$$

∴ 项目[B→•ξ, b]对γ = δα是有效的

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

$S' \overset{*}{\Rightarrow}_R \delta A\omega \Rightarrow_R \delta\alpha\beta\omega$

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

- 构造LR(1)项目集规范族
  - 闭包函数CLOSURE
  - 转换函数GO

项目集的闭包CLOSURE

- 假定I是文法G′的任一项目集，定义和构造I的闭包CLOSURE(I)如下：

  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(βa) 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

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

LR(1)项目集规范族的构造算法

```
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

- 把G拓广为G′
- 对G′构造LR(1)项目集规范族C和活前缀识别自动机的状态转换函数GO
- 使用C和GO，构造LR(1)分析表
  - 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
  - 构造LR(1)分析表的ACTION和GOTO子表

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)和SLR(1)分析表构造方法的对比

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”。

SLR(1)分析表的ACTION和GOTO子表构造

1. 若项目A→α•aβ属于I_k且GO(I_k,a)=I_j，a为终结符，则置ACTION[k,a] “sj” ；
2. 若项目A→α•属于I_k，那么，对任何终结符a∈FOLLOW(A)，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，A为非终结符，则置GOTO[k,A]=j；
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。

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。
- 具有规范的LR(1)分析表的文法称为一个LR(1)文法。
- 使用LR(1)分析表的分析器叫做一个规范的LR分析器。

LR(1)分析表和LR(1)文法

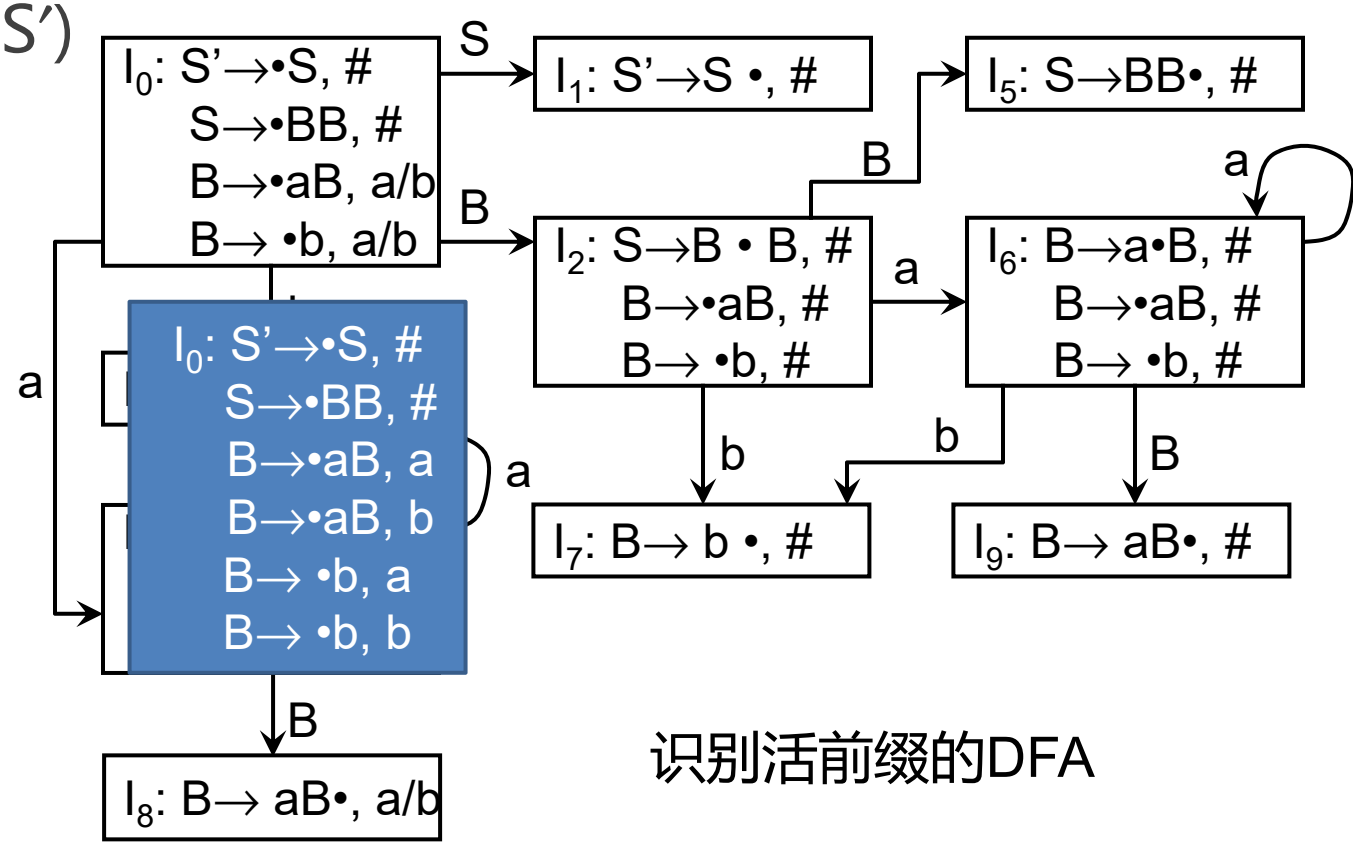
- 按上述算法构造的分析表，若不存在多重定义的入口(即，动作冲突)的情形，则称它是文法G的一张规范的LR(1)分析表。
- 具有规范的LR(1)分析表的文法称为一个LR(1)文法。
- 使用LR(1)分析表的分析器叫做一个规范的LR分析器。
- LR(1)状态比SLR(1)多
- LR(0) ⊂ SLR(1) ⊂ LR(1) ⊂ 无二义文法

编译原理

LR(1)分析表构造示例

示例：LR(1)分析表的构造

- 拓广文法G(S′)
  - (0) S′→S
  - (1) S→BB
  - (2) B→aB
  - (3) B→b

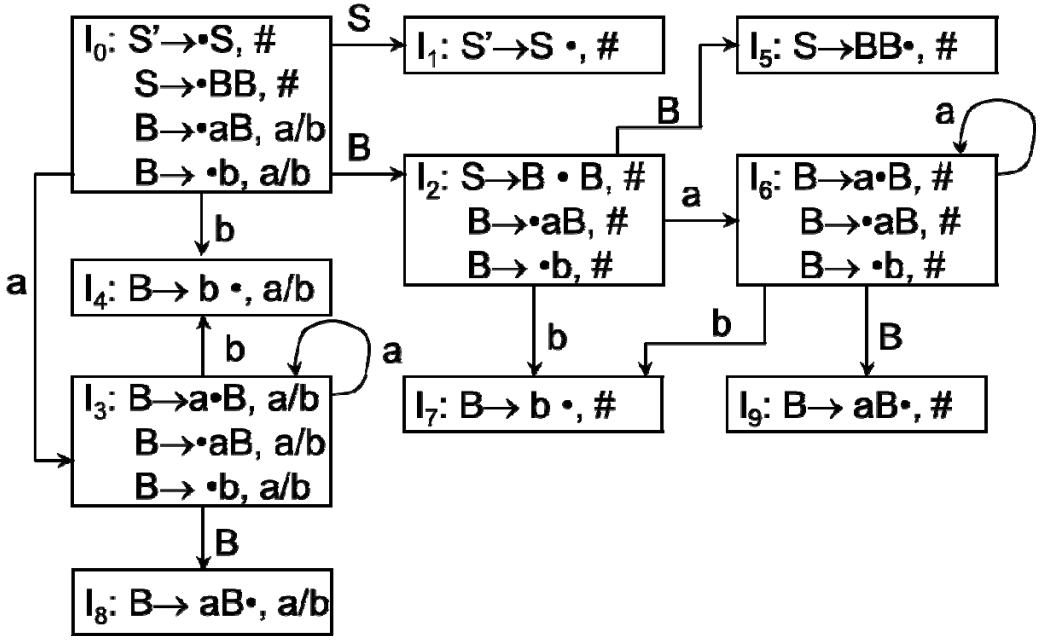


识别活前缀的DFA

示例：LR(1)分析表的构造

(0) S′→S
(1) S→BB
(2) B→aB
(3) B→b

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



LR(1)分析示例

(0) S′→S
(1) S→BB
(2) B→aB
(3) B→b

- 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

	ACTION			GOTO	
状态	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		

编译原理

分析器产生工具

分析器产生器——YACC

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

小结

- SLR(1)分析
  - 构造LR(0)项目集规范族、识别活前缀的DFA、
  - SLR(1)冲突解决办法
  - 构造SLR(1)分析表
- 规范LR(1)分析
  - 构造识别活前缀的DFA、LR(1)项目集规范族
  - LR(1)分析表的构造