Análise Sintática
(Cap. 04)
Analisador Sintático LR canônico e LALR

Análise Sintática LR

- O método LR (canônico LR) usa símbolos de *lookahead* para tomar decisões de shift-reduce
- O método LALR (LR com lookahead) é baseado no analisador LR canônico mas utiliza uma técnica para compactar os estados
- Para entender, considere a gramática:
  - S-> L = R | R
  - L-> *R | id
  - R -> L
- Que produz, dentre outros, os seguintes conjuntos de itens:

I_0

S'-> .S
S-> .L = R
S-> .R
L-> .*R
L-> .ID
R-> .L

I_2

S-> L . = R
R-> L .

Observe que AÇÃO[2,=] = "reduce R->L"

Análise Sintática LR

- Para entender, considere a gramática:
  - S-> L = R | R
  - L-> *R | id
  - R -> L
- Que produz, dentre outros, os seguintes conjuntos de itens:

I_0

S'-> .S
S-> .L = R
S-> .R
L-> .*R
L-> .ID
R-> .L

I_2

S-> L . = R
R-> L .

Observe que AÇÃO[2,=]= “reduce R->L”

Mas, não existe forma setencial à direita que inicia com R= ...

Análise Sintática LR

- Adicionaremos mais informações em um estado de maneira a evitar decisões “erradas” de shift-reduce
- Esse novo estado carregará, em cada item, a informação do próximo símbolo terminal válido para que um handle possa fazer um reduce
- A forma geral de um item LR passa a ser:
  - [A->C.D,a], onde **A->CD** é uma produção e **a** é um símbolo terminal ou o marcador $

Análise Sintática LR

- A construção de conjuntos de itens é quase a mesma que os algoritmos apresentados para o SLR

```
SetOfItems FECHO(I){
    repeat
      for (cada item [A->C.BD,a] em I)
        for (cada produção B->y em G')
          for (cada terminal b em FIRST(Da))
            add [B->.y,b] para I;
    até que itens não sejam mais adicionados para I;
    return I;
}
```

```
SetOfItems GOTO(I,X){
    iniciar J como conjunto vazio;
    for (cada item [A->C.XD,a] em I)
      add item [A-> CX.D,a] para J;
    return FECHO(J);
}
```

```
void items(G'){
    iniciar C=FECHO({[S'->.S,$]});
    repeat
      for (cada conjunto I em C)
        for (cada símbolo X da gramática)
          if (GOTO(I,X)!=vazio e não está em C)
            add GOTO(I,X) para C
    até que nenhum novo conjunto seja adicionado para C
}
```

Análise Sintática LR

- Considere a gramática:
  - S'-> S
  - S-> CC
  - C-> cC | d
- Após aplicar o algoritmo FECHO para I={[S'->.S,$]}, temos o conjunto I_0:

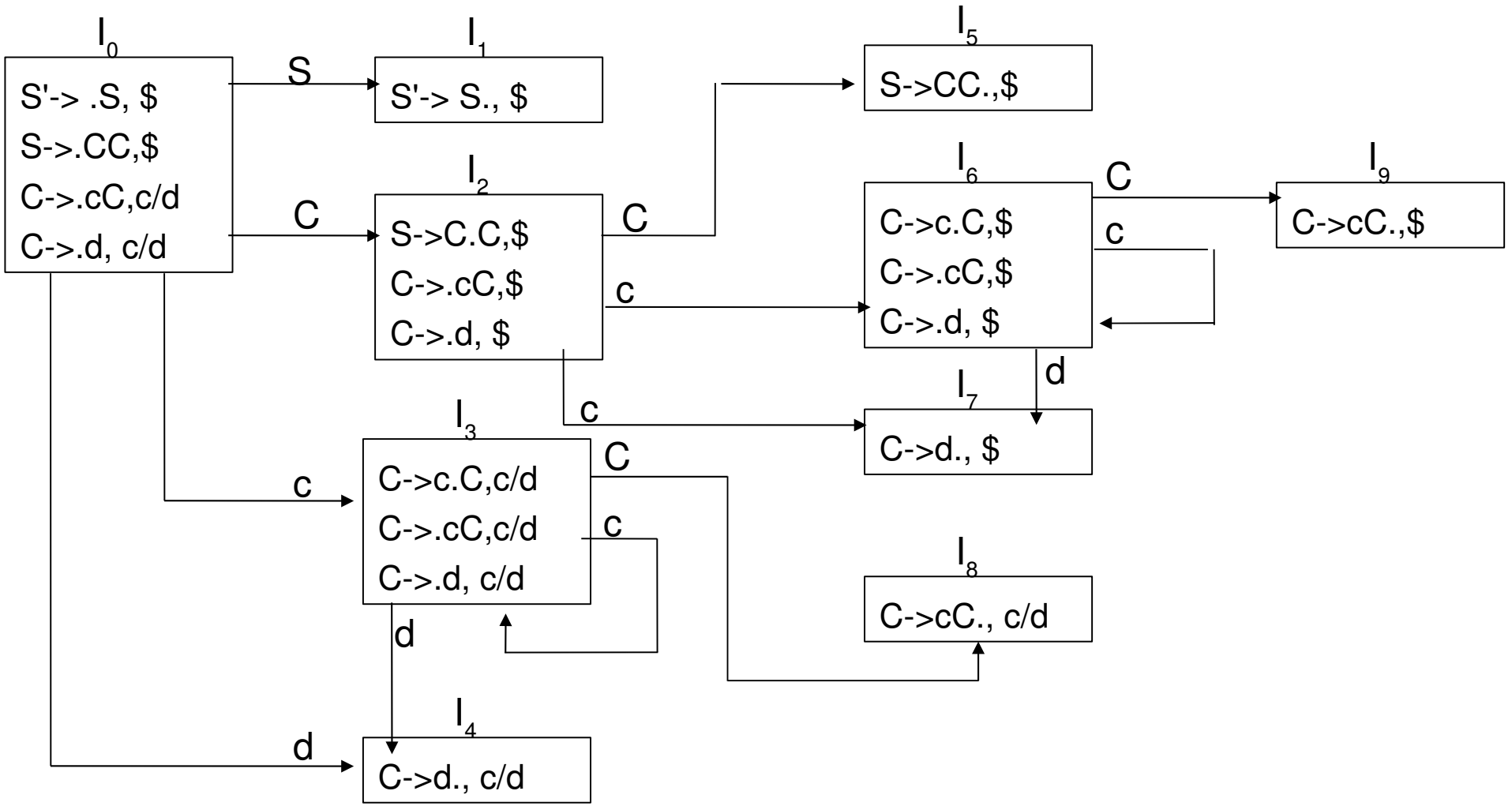
I_0

S'-> .S, $
S->.CC,$
C->.cC,c/d
C->.d, c/d

- A aplicação dos algoritmos items, FECHO e GOTO gera o seguinte autômato:

Análise Sintática LR

- Considere a gramática:
  - S'-> S
  - S-> CC
  - C-> cC | d



Análise Sintática LR

- As entradas AÇÃO e GOTO são construídas a partir do algoritmo
- ENTRADA: gramática G'

1. Construir C={I_0, I_1, ..., I_n}, a coleção de itens LR(1) para G'

2. Estado *i* é construído de I_i. As ações para o estado *i* são determinadas como segue:

a) Se [A->C.aD,b] está em I_i e GOTO(I_i,a)=I_j, então determinar AÇÃO[i,a]= shift j. a deve ser um terminal

b) Se [A->C.,a] está em I_i, A!=S' então AÇÃO[i,a]=reduce A->C.

c) Se [S'->S.,$] está em I_i, então AÇÃO[i,$]=accept

3. As transições GOTO são construídas a partir dos não-terminais usando a regra: if GOTO(I_i,A)=I_j, então GOTO[i,A]=j

4. Todas as entradas não definidas através dos passos 2) e 3) indicam “erro”

5. O estado inicial do parser é aquele do conjunto de itens [S'->.S,$]

Análise Sintática LR

- **Tabela de Análise Sintática LR canônica**
  - S'->S, 1) S-> CC, 2) C-> cC, 3) C-> d

Estado	AÇÃO			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		

Análise Sintática LALR

- Tabelas obtidas por este método são menores que as tabelas de um analisador LR canônico além de conseguir representar muitas construções de LP
- Considerando a gramática LR(1) anterior, observe que os estados 4 e 7 têm itens cujo primeiro componente (o item LR(0)) é igual (Ex: C->d.)
- Dessa forma, pode-se considerar que esses estados possuem itens com o mesmo *core* e, portanto, podem ser unidos para minimizar espaços nas tabelas AÇÃO e GOTO
- Existem outros conjuntos de itens que podem ser unidos?
- Como garantir que a junção de itens não introduzirá conflitos shift/reduce?

Análise Sintática LALR

- ***Para uma gramática LR(1) (é livre de conflito shift/reduce), a união resultante de itens com mesmo core não gerará um novo conflito***

- **Demonstração:** Supor uma gramática LR(1) e que a união de dois conjuntos de itens (estados) gera um conflito no símbolo lookahead *a* pois existe um item [A->B.,a] que executará uma redução “A->B” e existe um outro item [C->K.aY,b] que executa um shift. Então, algum conjunto de itens I que faz parte da união, possui item [A->B.,a] e, desde que o *core* dos conjuntos devem ser os mesmos, deve possuir o item [C->K.aY,c]. Logo, o conjunto I possui um conflito shift/reduce e, portanto, a gramática não é LR(1).

Análise Sintática LALR

- **Algoritmo para construção da tabela LALR**
- ENTRADA: gramática G'
- SAÍDA: TABELA LALR

1. Construir C={I_0, I_1, ..., I_n}, a coleção de itens LR(1)

2. Encontrar itens de LR(1) com o mesmo *core* e uni-los

3. Seja C'={J_0, J_1, ..., J_m} os conjuntos resultantes do passo 2. O conteúdo da tabela AÇÃO para o estado *i* é definido de J*i* da mesma forma que no algoritmo para construção da tabela LR(1). Se existir um conflito, o algoritmo falha e a gramática não é LALR(1)

4. A tabela GOTO é construída a partir de J, J=resultado da união entre os conjuntos de itens I_1, I_2, ..., I_k. Então, os *cores* dos GOTO(I_1,X), GOTO(I_2,X),..., GOTO(I_k,X) são os mesmos. Seja K=GOTO(J,X)

Análise Sintática LALR

- **Exemplo – Tabela de análise LALR**
  - S'->S, 1) S-> CC, 2) C-> cC, 3) C-> d

Estado	AÇÃO			GOTO	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		