Análise Sintática
(Cap. 04)
Análise Sintática Ascendente
Analisador Sintático LR

Análise Sintática LR

- Analisadores sintáticos LR(*k*):
  - L, verificação da entrada da esquerda para direita
  - R, constrói a derivação reversa mais a direita
  - *k*, número de símbolos da entrada que são usados para tomar decisão
  - SLR (Simple LR): método para construção de analisadores sintáticos shift-reduce
- Analisadores LR utilizam uma tabela para tomar decisões da análise sintática
- Analisadores LR podem ser utilizados para reconhecer uma grande quantidade de construções de LP
- O método utilizado por um analisador LR é o mais conhecido método que não utiliza *backtrackings* para shift-reduce
- Principal desvantagem: muito esforço para construir um analisador LR “do zero”.

Análise Sintática LR

- Como um analisador *shift-reduce* sabe quando realizar o shift ou o reduce?
- Um analisador LR toma decisões de *shift-reduce* através de estados em cada "momento" da análise
  - Estados representam conjuntos de itens
- Um item de uma gramática G é uma produção de G com um ponto em alguma posição no corpo da produção

  Produção A->XYZ possui quatro itens:

  A->.XYZ, A->X.YZ, A->XY.Z e A->XYZ.

  Um item indica "quanto" já foi analisado de uma produção!

Análise Sintática LR

- Uma coleção de itens LR é chamado de **coleção canônica LR**
- Essa coleção é a base para a construção de um AFD que é usado nas decisões de análise sintática
- Cada estado do autômato representa um conjunto de itens
- Para construir a coleção de itens LR para uma gramática, deve-se definir uma gramática estendida e duas funções: **FECHO** (closure) e **GOTO**
  - Se G é uma gramática com símbolo inicial S, então G' é a gramática estendida tal que S'->S, é o símbolo inicial de G'
  - A aceitação de uma entrada ocorre apenas quando o analisador reduz de S para S'

Análise Sintática LR

- **FECHO** de conjuntos de itens
  - Se I é um conjunto de itens para uma gramática G, então FECHO(I) é o conjunto de itens construídos de I com as regras:

1. Inicialmente, adicionar todo item em I para FECHO(I)
2. Se A->a.Bb está em FECHO(I) e B->y é uma produção, então adicionar o item B->.y para FECHO(I), se já não está lá.

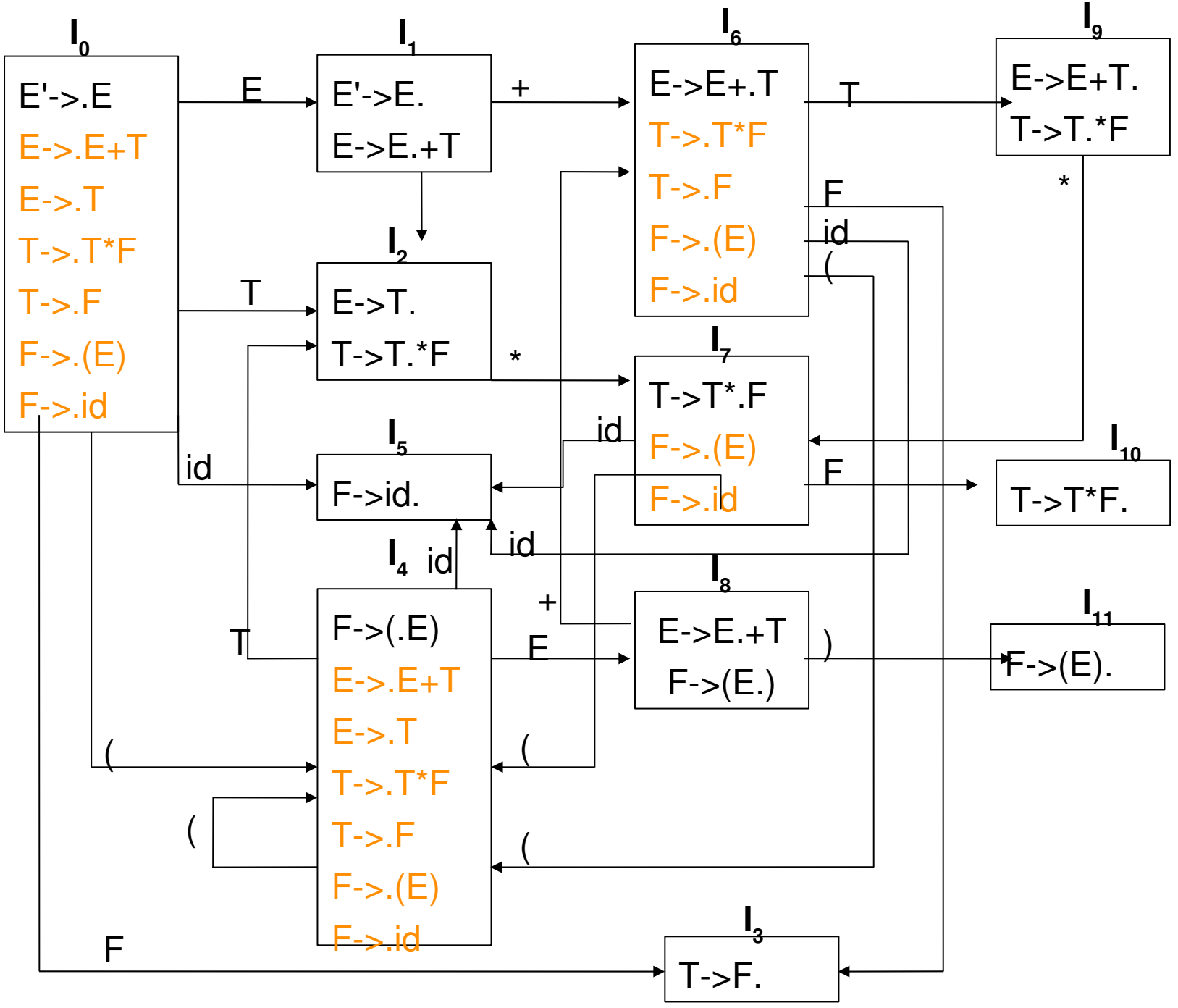
   OBS: Aplicar a regra 2 até não existirem itens que podem ser adicionados para o FECHO(I)

- A->a.Bb no FECHO(I) indica que, em algum momento, espera-se obter uma substring derivada de Bb na entrada
  - A substring derivada de Bb terá um prefixo derivado de B. Logo, adiciona-se itens para as produções de B. Ou seja, se B->y, então, B->.y está em FECHO(I).

Análise Sintática LR

- Analisador LR - Exemplo
  - Considere a gramática:
    - E'-> E
    - E -> E + T | T
    - T -> T * F | F
    - F -> (E) | id
  - Se I é o conjunto com um item {E'->.E}, então, Fecho(I) contém o conjunto de itens I_0 a seguir:

Analisador Sintático LR



Análise Sintática LR

- A função FECHO pode ser computada conforme o algoritmo a seguir:

```
SetOfItems FECHO(I){
    J=I;
    repeat
       for (cada item A->a.Bb em J)
          for (cada produção B->y de G)
             if (B->.y não está em J)
                add B->.y em J
    until não existem mais itens para adicionar em J
    return J;
}
```

- Note que se uma produção B é adicionada para o Fecho(I) com o ponto no início da produção, então todas as produções de B serão adicionadas para o fecho

Análise Sintática LR

- Os conjuntos de itens de interesse são divididos em:
  - **Itens de kernel**: o item inicial (S'->S) e todos os itens cujos pontos não estão no início da regra de produção
  - **Itens não-kernels**: todos os itens com pontos no início da regra de produção, exceto o item inicial

- Cada conjunto de itens de interesse é formado pelo FECHO de um conjunto de itens de kernel. Assim, os itens adicionados para o FECHO não são itens de kernel

- Pode-se representar os itens de interesse com menor espaço de armazenamento se ignorar todos os itens não-kernels

  - Itens não-kernels são representados na cor laranja, na figura anterior

Análise Sintática LR

- A função Goto(I,X) é definida como o fecho do conjunto de todos os itens A->aX.b tal que A->a.Xb está em I
- A função Goto possibilita definir as transições no autômato LR
- Os estados do autômato correspondem ao conjunto de itens Goto(I,X) indica a transição de I sob a entrada X
  - I é um conjunto de itens e X é um símbolo da gramática
- Exemplo: verifique novamente a figura do autômato LR e, dado o conjunto de itens {E'->E., E->E.+T}, então, o Goto(I,+) contém:
- E->E+.T
- T->.T*F
- T->.F
- F->.(E)
- F->.id

Para computar Goto(I,+), examina-se I nos itens que possuem + imediatamente à direita do ponto. Move-se então o ponto após o + e calcula-se o fecho desse item.

Análise Sintática LR

- Algoritmo para computar a coleção canônica de itens de LR para uma gramática G'

```
void items(G'){
    C=Fecho({S'->.S}); //Observe que S'->S é o símbolo inicial de G'
     repeat
        for (cada conjunto de itens I em C)
           for (cada símbolo da gramática X)
              if (Goto(I,X) não está vazio e não está em C)
                 add Goto(I,X) em C
     until não exista conjuntos para serem adicionados  em C
}
```

  - Exercício: Considere a gramática a seguir e aplique o algoritmo para construção da coleção de itens:
    - E'-> E, E -> E + T | T, T -> T * F | F, F -> (E) | id

Análise Sintática LR

- A idéia por trás do analisador SLR é a análise a partir do autômato LR(0).
  - Nesse autômato, os estados são conjuntos de itens da coleção canônica e as transições são dadas pela função **GOTO**
- O estado inicial do autômato LR é dado pelo fecho do estado inicial de G'. Um estado *j* refere-se ao estado correspondendo ao conjunto de itens I_j
- As decisões de **shift-reduce** são feitas da seguinte maneira:
  1. Supor que uma string w de símbolos de G' inicia o autômato LR com uma transição do estado inicial 0 para o estado *j*.
  2. Então, desloca-se (shift) o próximo símbolo da entrada *a* se *j* tem uma transição com *a.*
  3. Se não existe tal transição, faz-se a redução (reduce). Os itens que compõem *j* indicam qual produção usar na redução.

Análise Sintática LR

- **Exemplo:**
  - Considere a gramática anterior e a entrada id*id. Utiliza-se uma pilha para armazenar os estados

Pilha	**Símbolos**	**Entrada**	**Ação**
0	$	id*id$	shift para 5
0 5	$id	*id$	reduce para F->id
0 3	$F	*id$	reduce para T->F
0 2	$T	*id$	shift para 7
0 2 7	$T*	id$	shift para 5
0 2 7 5	$T*id	$	reduce para F->id
0 2 7 10	$T*F	$	reduce para T->T*F
0 2	$T	$	reduce para E->T
0 1	$E	$	accept

Análise Sintática LR

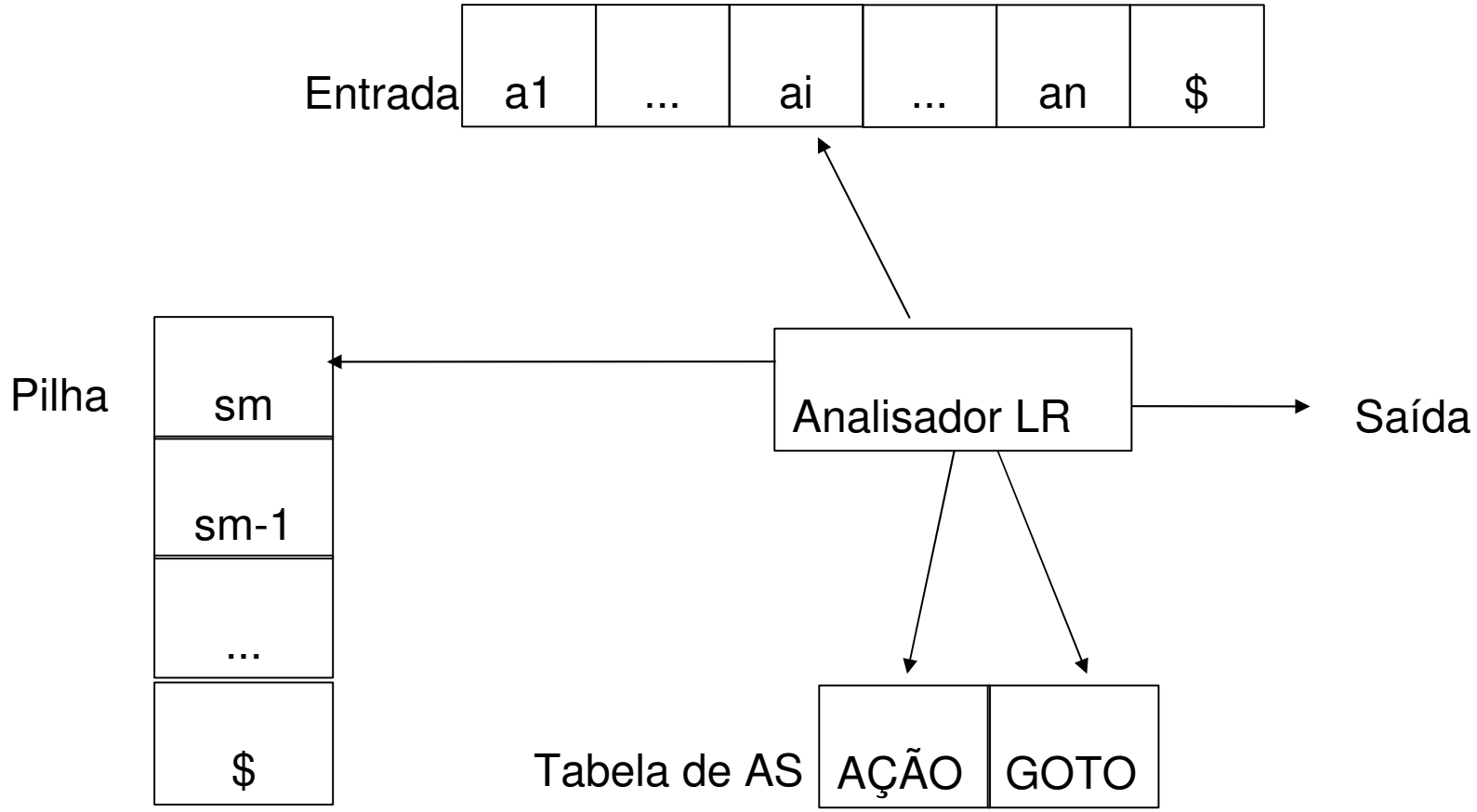
Pilha	Símbolos	Entrada	Ação
0	$	id*id$	shift para 5
0 5	$id	*id$	reduce para F->id
0 3	$F	*id$	reduce para T->F
0 2	$T	*id$	shift para 7
0 2 7	$T*	id$	shift para 5
0 2 7 5	$T*id	$	reduce para F->id
0 2 7 10	$T*F	$	reduce para T->T*F
0 2	$T	$	reduce para E->T
0 1	$E	$	accept

- Observe que a partir do estado 0 há uma transição com **id** (primeiro símbolo da entrada) para o estado 5 (shift).

- No estado 5, não existe transição com o símbolo *, portanto, deve-se reduzir **F->id.** Note que a redução consiste em: trocar id pela cabeça da produção (F) e retirar o estado 5 (id), retornar para estado 0 (estado anterior) e procurar uma transição com F (estado 3)

Análise Sintática LR

- Algoritmo para Análise Sintática LR



Pilha mantém uma seqüência de estados. Cada estado, exceto o estado inicial, possui somente um único símbolo da gramática associado a si.

Análise Sintática LR

- A Tabela de análise sintática consiste de duas funções: AÇÃO e GOTO
  - A função AÇÃO recebe como argumento um estado ***i*** e um terminal ***a***. O retorno de **AÇÃO[*i,a*]** pode ser:

    a) Deslocar *j*, onde *j* é um estado. Desloca *a* para a pilha. Use estado *j* para representar *a*
    b) Reduz A->b. Reduz b, no topo da pilha, para A
    c) Aceitar. Analisador aceita a entrada e finaliza
    d) Erro. Dependendo do erro, pode finalizar ou executar alguma ação

  - Estende-se a função GOTO para estados: se **GOTO[I_i,A]=I_j**, então GOTO mapeia o estado ***i*** e um não-terminal **A** para o estado ***j***.

Análise Sintática LR

- O comportamento do analisador sintático SLR é então definido da seguinte maneira:
  - Lê **a**, o símbolo atual da entrada, e **s**, o estado atual (no topo da pilha) e consulta a entrada AÇÃO[s,a] na Tabela de Análise Sintática.
  - O retorno de AÇÃO[s,a] deve ser um dos quatro possíveis: deslocar (shift) para estado s, reduzir, aceitar ou erro.

- A partir da definição de AÇÃO e GOTO, pode-se definir o algoritmo para um analisador sintático LR conforme a seguir

Análise Sintática LR

- **Algoritmo SLR**
- ENTRADA: string w e tabela com funções AÇÃO e GOTO para gramática G
- SAÍDA: se w está em L(G), aceitação para w, senão, erro!

```
Seja a, o primeiro símbolo de w$
while (1){
    Seja s o estado no topo da pilha
    if (AÇÃO[s,a]==shift t){
       push t;
       Seja a o próximo símbolo da entrada;
    }else if (AÇÃO[s,a]==reduce A->b){
        pop |b| símbolos da pilha;
        Seja t o estado no topo da pilha;
        push GOTO[t,A];
    }else if (AÇÃO[s,a]==aceitar) break;
    else erro();
}
```

Análise Sintática LR

- **Exemplo – algoritmo LR**
- Considere as regras de produção de G:
- 1) E-> E+T; 2) E-> T; 3) T-> T*F;
- 4) T-> F; 5) F-> (E); 6) F->id
- Os códigos para as ações são:
  - **si** significa shift e empilhar estado **i**
  - **rj** significa reduce para a produção **j**
  - **acc** significa accept
  - Espaço em branco significa erro

Análise Sintática LR

- **Exemplo – algoritmo LR e Tabela de Análise Sintática**

Estado	AÇÃO						GOTO		
	ID	+	*	(	)	$	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			

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 para G'

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

a) Se [A->c.ab] 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->B.] está em I_i, então AÇÃO[i,a]=reduce A->B, para todo a em FOLLOW(A); A não deve ser S'.

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]