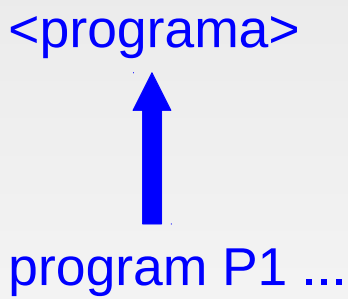
Construção de Compiladores

Análise Sintática Ascendente – parte2

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

Análise Sintática Ascendente

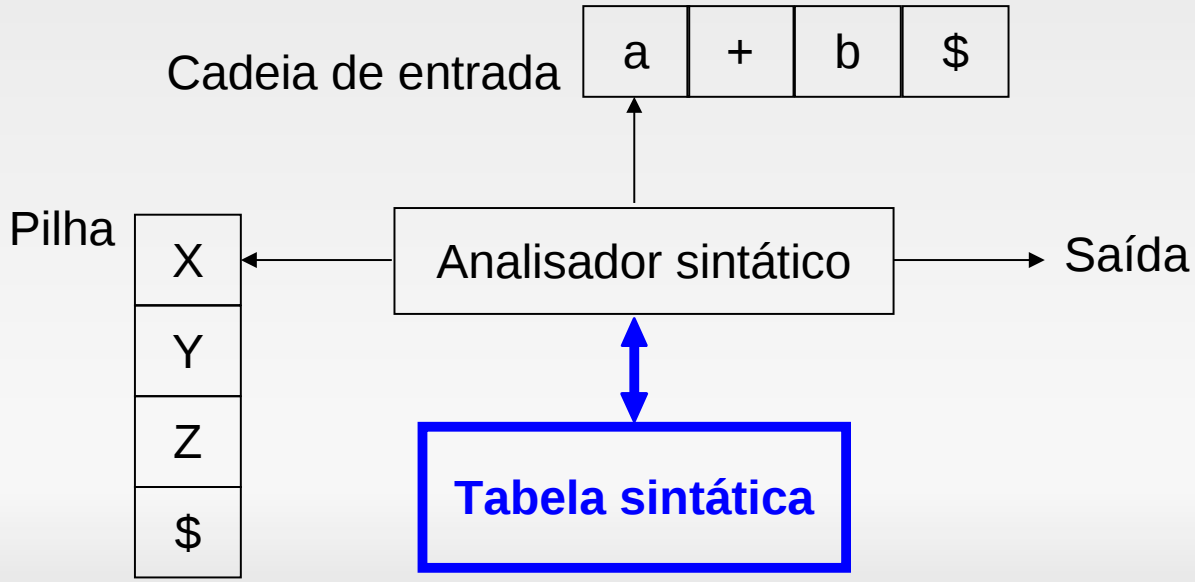
- Como é feita?
  - A análise é feita das folhas para a raiz
  - Parte-se das folhas (sequência de *tokens* retornada pelo analisador léxico) e, por meio de reduções, chega-se ao símbolo inicial da gramática
  - Analisadores de **empilha-reduz** (*shift-reduce*)



    - Redução
    - Derivação à direita

Análise Sintática Ascendente

- Como é feita?
  - Componentes
    - **Pilha** – onde os símbolos a serem reduzidos são empilhados
    - **Tabela sintática** – guia o processo de empilha/reduz

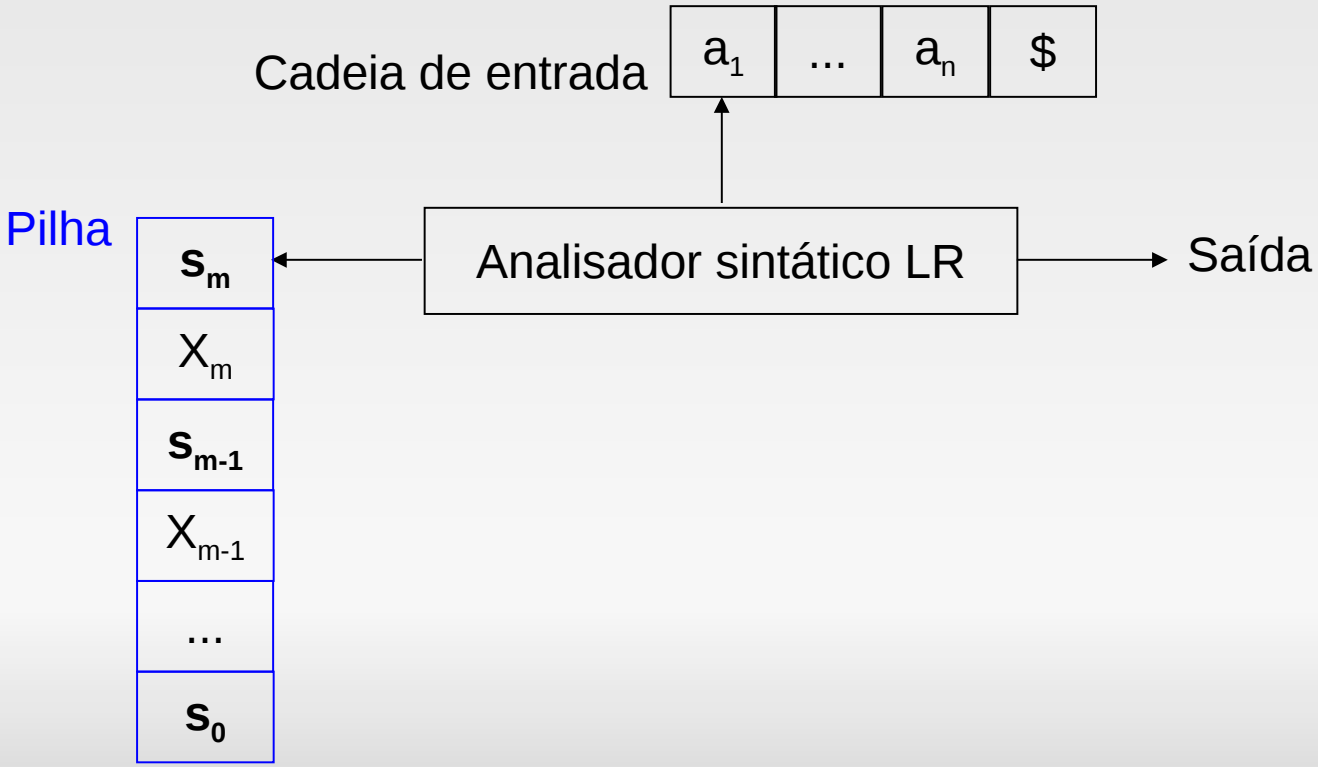


Análise Sintática Ascendente

- Analisadores sintáticos ascendentes (ASA) – 2 tipos
  - Analisador de precedência de operadores
    - Opera sobre a classe das gramáticas de operadores
    - Guiado por uma tabela de precedência
  - Analisador LR (*k*)
    - L*eft to right with* R*ightmost derivation*
      - Lê a sentença em análise da esquerda para a direita
      - Produz uma derivação mais à direita ao reverso
      - Considerando-se *k* símbolos na cadeia de entrada

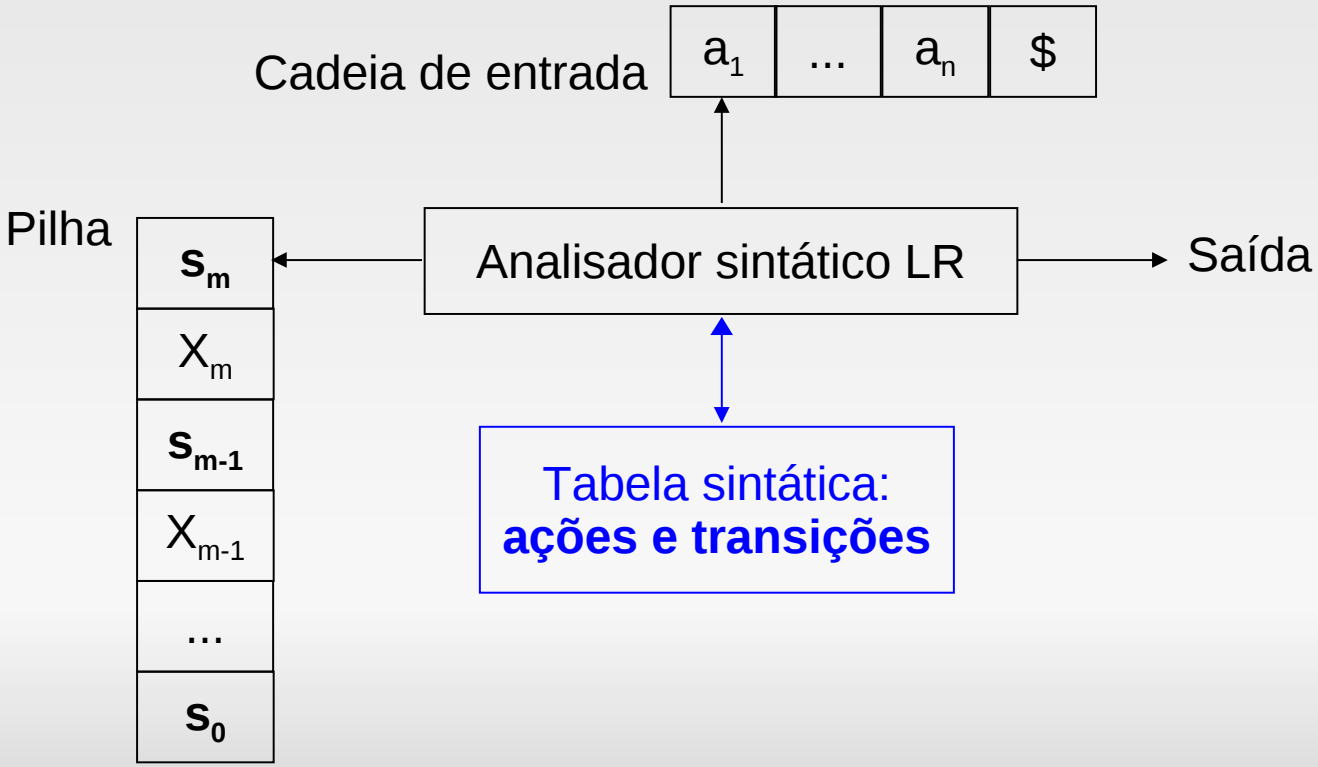
Análise Sintática Ascendente LR

- Como é feita?
  - Componentes
    - Pilha
      - X_i – símbolos gramaticais
      - s_i – estados (resumem a informação abaixo na pilha)



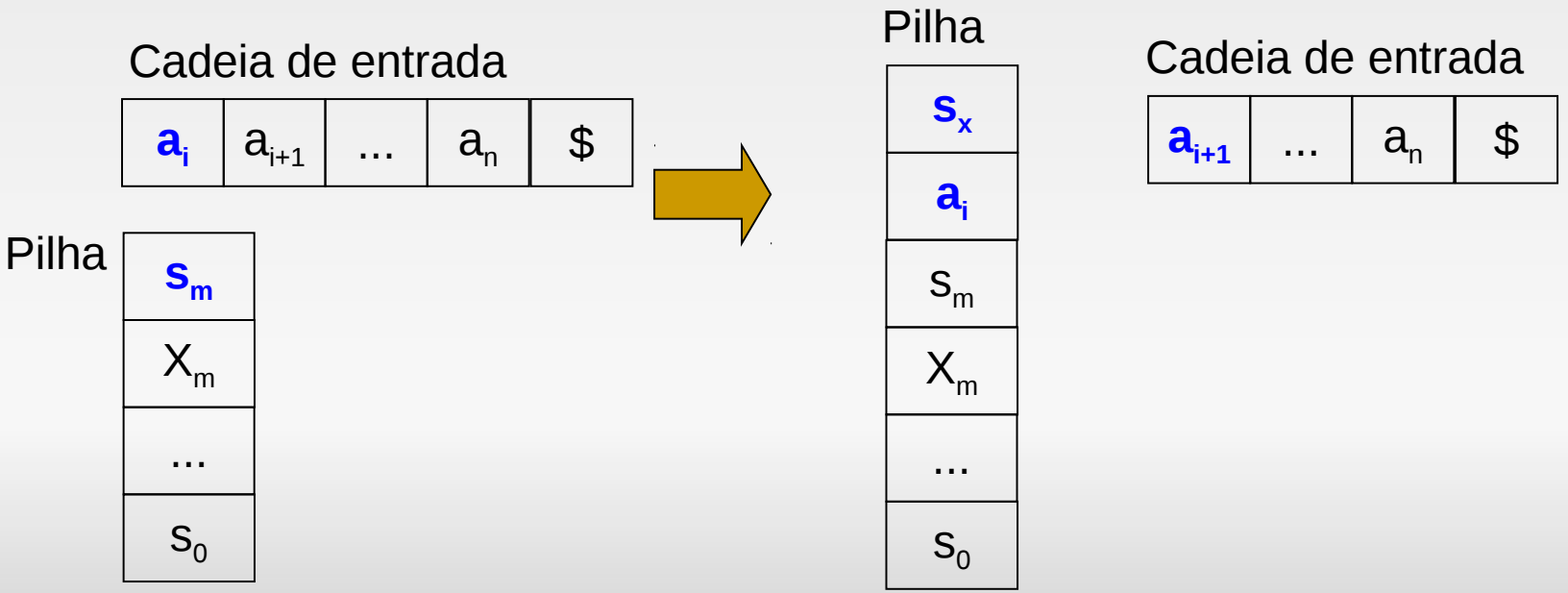
Análise Sintática Ascendente LR

- Como é feita?
  - Componentes
    - Tabela sintática
      - **Ações** (empilha ou reduz) associadas às transições de estados
      - **Transições** de estados relacionadas aos não-terminais



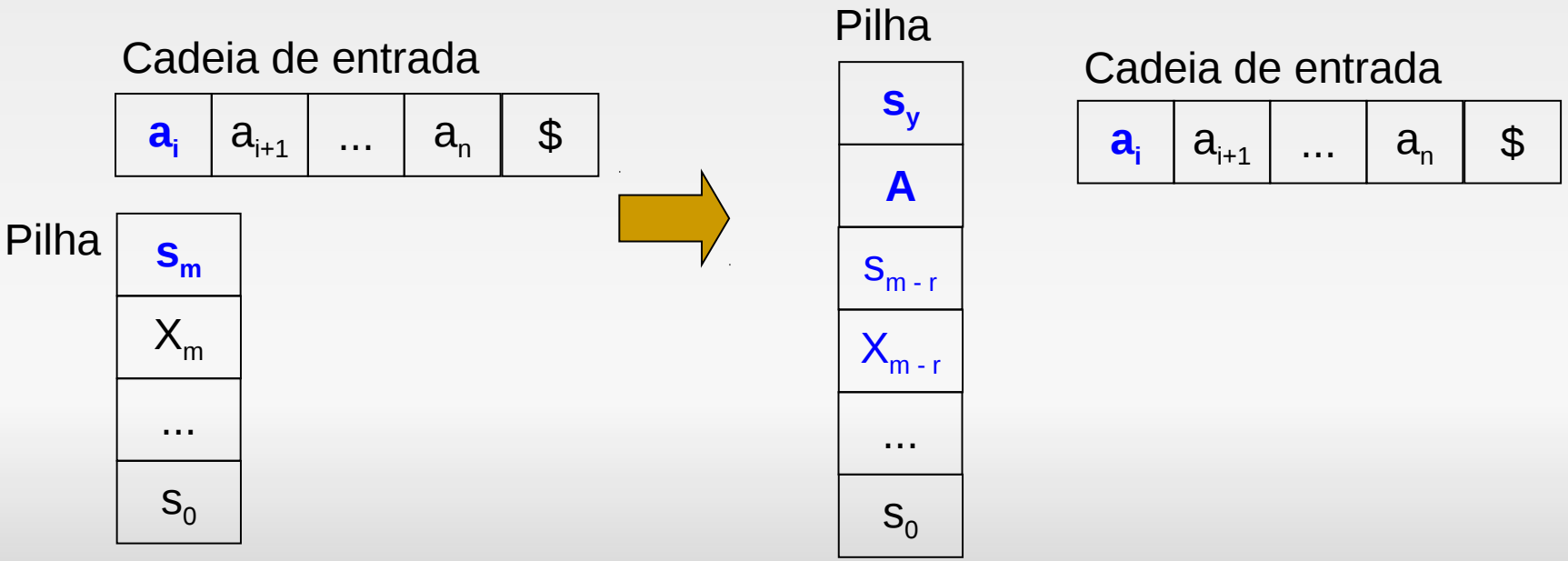
Análise Sintática Ascendente LR

- Como é feita?
  - Sejam
    - s_m o estado no topo da pilha
    - a_i o *token* sob o cabeçote de leitura
  - Tabela AÇÃO[s_m,a_i] pode assumir um dos valores
    - Empilha
      - Empilha "$a_i s_x$" (em que s_x é o novo estado dado por AÇÃO[s_m,a_i]) e
      - Avança na entrada



Análise Sintática Ascendente LR

- Como é feita?
  - Sejam
    - s_m o estado no topo da pilha
    - a_i o *token* sob o cabeçote de leitura
  - Tabela AÇÃO[s_m,a_i] pode assumir um dos valores
    - Reduz – usando a produção A →β
      - Desempilha 2r símbolos onde r = |β| e
      - Empilha "As_y" onde s_y resulta da tabela TRANSIÇÃO[s_{m-r},A]



Análise Sintática Ascendente LR

- Como é feita?
  - Sejam
    - s_m o estado no topo da pilha
    - a_i o *token* sob o cabeçote de leitura
  - Tabela AÇÃO[s_m,a_i] pode assumir um dos valores
    - Aceita
      - Reconhece a sentença de entrada como válida, pois é essa a ação especificada na tabela (ACEITA ou OK)

Análise Sintática Ascendente LR

- Como é feita?
  - Sejam
    - s_m o estado no topo da pilha
    - a_i o *token* sob o cabeçote de leitura
  - Tabela AÇÃO[s_m,a_i] pode assumir um dos valores
    - Erro
      - ➔ Para a execução indicando um erro sintático, pois não há ação/transição possível na tabela sintática para a configuração (pilha + entrada) atual

Análise Sintática Ascendente LR

- Exemplo
  - Gramática de expressões aritméticas
    (1) <E>::=<E>+<T>
    (2) <E>::=<T>
    (3) <T>::=<T>*<F>
    (4) <T>::=<F>
    (5) <F>::=(<E>)
    (6) <F>::=id

Tabela sintática LR

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
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			

Na tabela, tem-se que:
- s*i* indica “empilhar *i*”
- r*i* indica “reduzir por regra *i*”

Análise Sintática Ascendente LR

- Algoritmo de análise sintática LR

empilha o estado inicial s_0;
concatena o símbolo delimitador $ no final da cadeia de entrada;
faz ip apontar para o primeiro símbolo da cadeia;
do *(*seja* s_n *o estado no topo da pilha e a o símbolo apontado por ip *)*
  if (*ação*[s_n,*a*]=*"empilhar* s_{n+1}*"*) **then**
    empilha a; *(* empilha *)*
    empilha s_{n+1};
    avança ip; *(* avança na leitura da entrada*)*
  else if (*ação*[s_n,*a*]=*"reduzir A→ β"*) **then**
      desempilha 2|β| elementos*; *(* s está no topo da pilha *)*
      empilha A; *(* reduz *)*
      empilha o estado indicado por transição[$s_{n-|\beta|}$, *A*]; (*$s_{n-|\beta|}$ no topo*)
    else if (*ação*[s_n,*a*]=*"aceitar"*) **then** ACEITA
      else ERRO;
until ACEITA **or** ERRO;

Análise Sintática Ascendente LR

- Exemplo
  - Reconhecer a cadeia id*id+id

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
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			

Pilha	Cadeia	Ação
0	id*id+id$	

Análise Sintática Ascendente LR

- Exemplo
  - Reconhecer a cadeia id*id+id

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
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			

Pilha	Cadeia	Ação
0	id*id+id$	s5
0id5	*id+id$	r6
0F3	*id+id$	r4
0T2	*id+id$	s7
0T2*7	id+id$	s5
0T2*7id5	+id$	r6
0T2*7F10	+id$	r3
0T2	+id$	r2
0E1	+id$	s6
0E1+6	id$	s5
0E1+6id5	$	r6
0E1+6F3	$	r4
0E1+6T9	$	r1
0E1	$	**ACEITA**

Análise Sintática Ascendente LR

- Análise sintática ascendente LR (*k*) – 3 tipos
  - *Simple* LR (SLR)

    Fácil de implementar, mas menos poderoso
    - É fácil de implementar
    - Porém é aplicável a uma classe restrita de gramáticas
  - *Look Ahead* LR (LALR)

    Poder e complexidade intermediários
    - É o de nível intermediário e implementação eficiente
    - Funciona para a maioria das linguagens de programação
    - ➔ Yacc gera esse tipo de analisador
  - LR Canônico

    Mais complexo e mais poderoso
    - É o mais poderoso
    - Pode ser aplicado a um grande número de LLC
  - ➔ O programa diretor é o mesmo, o que muda é a tabela

ASA Simple LR

- Construindo a tabela sintática SLR
  - A construção da tabela SLR se baseia no conjunto canônico de itens LR(0)
    - LR(0): 0 porque não se olha nenhum símbolo a frente
  - Um item LR(0) para uma gramática G é uma produção com alguma indicação (.) de até onde essa produção já foi analisada no processo de reconhecimento
    - Exemplo – a produção A→XYZ dá origem a 4 itens LR(0):
      A→**.**XYZ
      A→X**.**YZ
      A→XY**.**Z
      A→XYZ**.**

      Itens como este, com **.** na última posição, são ditos completos
    - Produções do tipo A→ε geram somente um item A→**.**

      Itens como este, com **.** na última posição, são ditos completos

ASA Simple LR

- Construindo a tabela sintática SLR
  - Construção do conjunto canônico de itens LR(0)
    - Operações
      1) Acrescentar à gramática a produção S' →S (onde S é o símbolo inicial da gramática)
         - Permite a identificação do fim da análise, mais especificamente, com a aplicação de S'→S**.**
      2) Computar as funções fechamento e desvio para a nova gramática
    - Exemplo

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

ASA Simple LR

- Construindo a tabela sintática SLR
  - Construção do conjunto canônico de itens LR(0)
    - Operações

      1) Acrescentar à gramática a produção S' →S (onde S é o símbolo inicial da gramática)
         - Permite a identificação do fim da análise, mais especificamente, com a aplicação de S'→S**.**

      2) Computar as funções fechamento e desvio para a nova gramática

    - Exemplo

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

      Passo 1 – inserção do símbolo S':

      <S'>::= <S>

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

ASA Simple LR

- Construindo a tabela sintática SLR
  - Fechamento
    - Seja I um conjunto de itens LR(0)
      1. Todo item em I pertence ao fechamento(I)
      2. Se A→α.Xβ está em fechamento(I) e X→γ é uma produção, então adiciona-se X→.γ ao conjunto

    - Exemplo

      Dada a gramática:

      <S'>::= <S>

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

      e I = {<S> ::= [**.** <L>] }

ASA Simple LR

- Construindo a tabela sintática SLR
  - Fechamento
    - Seja I um conjunto de itens LR(0)
      1. Todo item em I pertence ao fechamento(I)
      2. Se A→α.Xβ está em fechamento(I) e X→γ é uma produção, então adiciona-se X→.γ ao conjunto

    - Exemplo

      Dada a gramática:

      <S'>::= <S>

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

      e I = {<S> ::= [**.** <L>] }

      Fechamento(I) = {<S> ::= [**.** <L>]}

Passo 1 – insere elemento de I em fechamento(I)

ASA Simple LR

- Construindo a tabela sintática SLR
  - Fechamento
    - Seja I um conjunto de itens LR(0)
      1. Todo item em I pertence ao fechamento(I)
      2. Se A→α.Xβ está em fechamento(I) e X→γ é uma produção, então adiciona-se X→.γ ao conjunto

    - Exemplo

      Dada a gramática:

      <S'>::= <S>

      <S>::= a | [<L>]

      <L>::= <L> ; <S> | <S>

      e I = {<S> ::= [**.** <L>] }

      Fechamento(I) = {<S> ::= [**.** <L>], <L>::= **.** <L> ; <S>, <L> ::= **.**<S>, <S>::= **.**a, <S> ::= **.** [<L>] }

      Passo 2 – para cada regra no conjunto, são adicionadas as regras dos não-terminais que aparecem precedidos pelo indicador **.**

ASA Simple LR

- Construindo a tabela sintática SLR
  - Desvio
    - Seja I um conjunto de itens LR(0)
      - *desvio*(I,X): consiste em
        1. Avançar o indicador (**.**) através do símbolo gramatical X (terminal ou não-terminal) das produções correspondentes em I e
        2. Calcular a função *fechamento* para o novo conjunto
    - Exemplo

      Considerando-se o conjunto

      I = {<S>::= [<L> **.**], <L>::=<L> **.** ; <S> }

      Desvio(I,;) = { <L>::=<L> ; **.** <S> }

      Passo 1 – avança o indicador **.** através de ;

ASA Simple LR

- Construindo a tabela sintática SLR
  - Desvio
    - Seja I um conjunto de itens LR(0)
      - *desvio*(I,X): consiste em
        1. Avançar o indicador (**.**) através do símbolo gramatical X (terminal ou não-terminal) das produções correspondentes em I e
        2. Calcular a função *fechamento* para o novo conjunto

    - Exemplo

      Considerando-se o conjunto

      I = {<S>::= [<L> **.**], <L>::=<L> **.** ; <S> }

      Desvio(I,;) = { <L>::=<L> ; **.** <S> , <S> ::= **.** a, <S> ::= **.** [<L>] }

      Passo 2 – calcula o fechamento para o novo conjunto

ASA Simple LR

- Algoritmo para obter o conjunto canônico de itens

C := { I_0 = fechamento({S'→ .S}) }

repita

para cada conjunto I em C e X símbolo de G, tal que desvio(I,X) ≠ ∅

adicione desvio(I,X) a C

até que todos os conjuntos tenham sido adicionados a C

- Exemplo

Dada a gramática:

<S'>::= <S>

<S>::= a | [<L>]

<L>::= <L> ; <S> | <S>

I_0 = **?**

ASA Simple LR

- Algoritmo para obter o conjunto canônico de itens

C := { I_0 = fechamento({S'→ .S}) }

repita

para cada conjunto I em C e X símbolo de G, tal que desvio(I,X) ≠ ∅

adicione desvio(I,X) a C

até que todos os conjuntos tenham sido adicionados a C

- Exemplo

Dada a gramática:

<S'>::= <S>

<S>::= a | [<L>]

<L>::= <L> ; <S> | <S>

I_0 = fechamento({<S'> ::= **.** <S>}) = {<S'> ::= **.** <S>, <S>::= **.** a,

<S> ::= **.** [<L>] }

ASA Simple LR

- Algoritmo para obter o conjunto canônico de itens

C := { I_0 = fechamento({S'→ .S}) }

repita

para cada conjunto I em C e X símbolo de G, tal que desvio(I,X) ≠ ∅

adicione desvio(I,X) a C

até que todos os conjuntos tenham sido adicionados a C

- Exemplo

Dada a gramática:

<S'>::= <S>

<S>::= a | [<L>]

<L>::= <L> ; <S> | <S>

I_0 = {<S'> ::= **.** <S>, <S>::= **.** a, <S> ::= **.** [<L>] }

desvio(I_0,S) = {<S'> ::= <S>**.**} = I_1

desvio(I_0,a) = {<S>::= a**.**} = I_2

desvio(I_0,[ ) = {<S> ::= [**.** <L>], <L>::= **.**<L> ; <S>, <L> ::= **.**<S>,

<S>::= **.** a, <S> ::= **.** [<L>] } = I_3

ASA Simple LR

- Algoritmo para obter o conjunto canônico de itens

C := { I_0 = fechamento({S'→ .S}) }

repita

para cada conjunto I em C e X símbolo de G, tal que desvio(I,X) ≠ ∅

adicione desvio(I,X) a C

até que todos os conjuntos tenham sido adicionados a C

- Exemplo

Dada a gramática:

<S'>::= <S>

<S>::= a | [<L>]

<L>::= <L> ; <S> | <S>

...

C = { {<S'> ::= . <S>, <S>::= . a, <S> ::= . [<L>] }, { <S'> ::= <S>. },
{ <S>::= a . }, { <S> ::= [. <L>], <L>::= .<L> ; <S>, <L> ::= .<S>,
<S>::= . a, <S> ::= . [<L>] }, { <S> ::= [<L> .], <L>::= <L>. ; <S> },
{ <L> ::= <S> . }, { <S> ::= [<L>] . }, { <L>::= <L> ; . <S>, <S>::= . a,
<S> ::= . [<L>] }, { <L>::= <L> ; <S> . } }

ASA Simple LR

- Algoritmo para construir a tabela de análise SLR

(**Após a construção do conjunto canônico C como descrito anteriormente, as tabelas AÇÃO e TRANSIÇÃO são construídas conforme esse algoritmo no qual a representa um terminal e A um não-terminal* *)*
Seja C={I_0, I_1, ..., I_n}, os **estados** do analisador são 0 (inicial), 1, ...n (**estados**)
A linha i da tabela é construída pelo conjunto I_i como segue: (**linhas**)
Ações na tabela (**estado X terminal**)
Se desvio(I_i,a)=I_j, **então** AÇÃO[i,a]=s*j*
Com exceção da regra <S'>::= <S> adicionada, para todas as outras regras,
Se <A>::=α**.** está em I_i, **então**, para todo a ∈ Seguidor(A), faça
AÇÃO[i,a]=reduz *n*, em que *n* é o número da produção <A>::=α
Se <S'>::=<S>**.** está em I_i, **então** faça AÇÃO[i,$]=ACEITA
Transições na tabela (**estado X não-terminal**)
Se desvio(I_i,A)=I_j, **então** TRANSIÇÃO(i,A)=j

- Entradas não definidas indicam ERRO
- Ações conflitantes indicam que a gramática não é SLR

ASA Simple LR

- Exemplo

  <S'>::= <S>
  <S>::= a | [<L>]
  <L>::= <L> ; <S> | <S>

  C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0							
1							
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

<S'>::= <S>
<S>::= a | [<L>]
<L>::= <L> ; <S> | <S>

C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
I_0= {<S'> ::= **.** <S>,
<S>::= **.** a,
<S> ::= **.** [<L>]
}
desvio(I_0,S) = {<S'> ::= <S>**.**}= I_1
desvio(I_0,a) = {<S>::= a**.**} = I_2
desvio(I_0,[ ) = {<S> ::= [**.** <L>],
<L>::= **.**<L> ; <S>, <L> ::= **.**<S>,
<S>::= **.** a, <S> ::= **.** [<L>] } = I_3

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0							
1							
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

<S'>::= <S>
<S>::= a | [<L>]
<L>::= <L> ; <S> | <S>

C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
I_0 = {<S'> ::= . <S>,
<S>::= . a,
<S> ::= . [<L>]
}
desvio(I_0,S) = {<S'> ::= <S>.}= I_1
desvio(I_0,a) = {<S>::= a.} = I_2
desvio(I_0,[ ) = {<S> ::= [. <L>],
<L>::= .<L> ; <S>, <L> ::= .<S>,
<S>::= . a, <S> ::= . [<L>] } = I_3

Se desvio(I_i,A)=I_j, então TRANSIÇÃO(i,A)=j

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0						1	
1							
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

<S'>::= <S>
<S>::= a | [<L>]
<L>::= <L> ; <S> | <S>

C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
I_0= {<S'> ::= . <S>,
<S>::= . a,
<S> ::= . [<L>]
}
desvio(I_0,S) = {<S'> ::= <S>.}= I_1
desvio(I_0,a) = {<S>::= a.} = I_2
desvio(I_0,[ ) = {<S> ::= [. <L>],
<L>::= .<L> ; <S>, <L> ::= .<S>,
<S>::= . a, <S> ::= . [<L>] } = I_3

Se desvio(I_i,a)=I_j, então AÇÃO[i,a]=s*j*

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2					1	
1							
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

<S'>::= <S>
<S>::= a | [<L>]
<L>::= <L> ; <S> | <S>

C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
I_0= {<S'> ::= . <S>,
<S>::= . a,
<S> ::= . [<L>]
}
desvio(I_0,S) = {<S'> ::= <S>.}= I_1
desvio(I_0,a) = {<S>::= a.} = I_2
desvio(I_0,[ ) = {<S> ::= [. <L>],
<L>::= .<L> ; <S>, <L> ::= .<S>,
<S>::= . a, <S> ::= . [<L>] } = I_3

Se desvio(I_i,a)=I_j, então AÇÃO[i,a]=s*j*

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1							
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

  <S'>::= <S>
  <S>::= a | [<L>]
  <L>::= <L> ; <S> | <S>

  C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
  I_1 = {<S'> ::= <S>.}

Se <S'>::=<S>. está em I_i, então faça AÇÃO[i,$]=ACEITA

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1					OK		
2							
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

  <S'>::= <S>
  <S>::= a | [<L>]
  <L>::= <L> ; <S> | <S>

  C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8}
  I_2= {<S>::= a.}

  Seguidor(S') = {$}
  Seguidor(L) = {], ;}
  Seguidor(S) = Seguidor(L) +
  Seguidor(S') = {$,], ; }

Se <A>::=α. está em I_i, então, para todo a em Seguidor(A), faça AÇÃO[i,a]=reduz *n*, em que *n* é o número da produção <A>::=α

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1					OK		
2			r1	r1	r1		
3							
4							
5							
6							
7							
8							

ASA Simple LR

- Exemplo

  <S'>::= <S>
  <S>::= a | [<L>]
  <L>::= <L> ; <S> | <S>

Aplicando-se o algoritmo de construção da tabela para todos os I_n tem-se a tabela final apresetada abaixo

Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1					OK		
2			r1	r1	r1		
3	s2	s3				5	4
4			s6	s7			
5			r4	r4			
6			r2	r2	r2		
7	s2	s3				8	
8			r3	r3			

ASA Simple LR

- Exemplo

  <S'>::= <S>
  <S>::= a | [<L>]
  <L>::= <L> ; <S> | <S>

Autômato correspondente

Como o autômato não é usado para reconhecer cadeias, mas para acompanhar o estado da análise sintática, não há estados finais

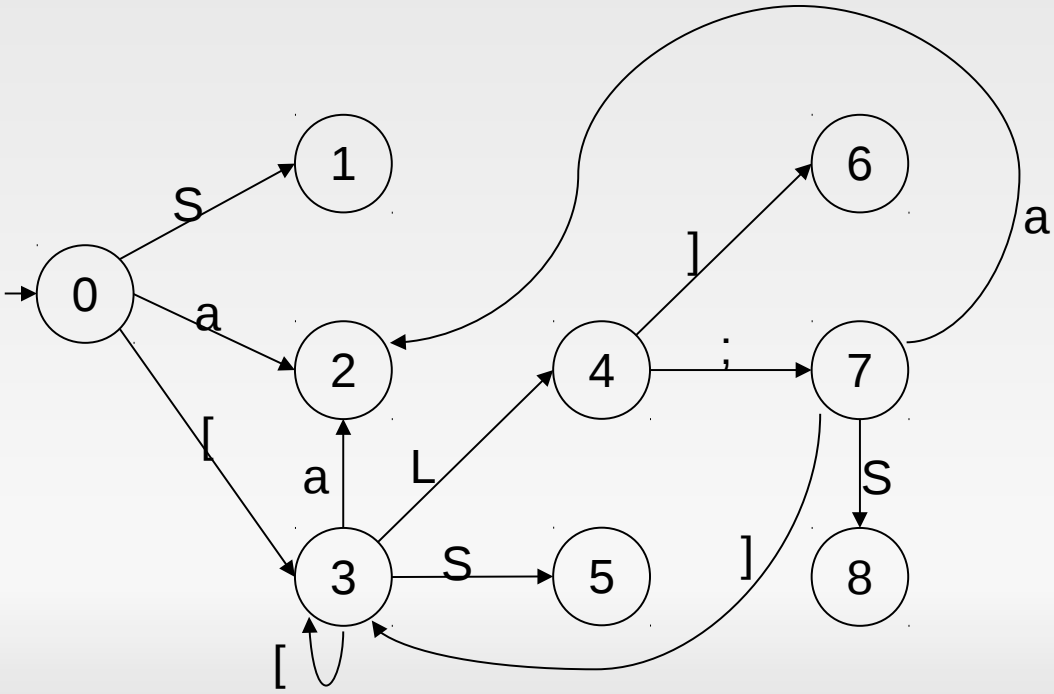


Tabela sintática SLR

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1					OK		
2			r1	r1	r1		
3	s2	s3				5	4
4			s6	s7			
5			r4	r4			
6			r2	r2	r2		
7	s2	s3				8	
8			r3	r3			

ASA Simple LR

- Exemplo - Reconhecer a cadeia [a ; a]

0. <S'>::= <S>
1. <S>::= a
2. <S>::= [<L>]
3. <L>::= <L> ; <S>
4. <L>::= <S>

Estados	Ações					Transições	
	a	[	]	;	$	S	L
0	s2	s3				1	
1					OK		
2			r1	r1	r1		
3	s2	s3				5	4
4			s6	s7			
5			r4	r4			
6			r2	r2	r2		
7	s2	s3				8	
8			r3	r3			

Pilha	Cadeia	Ação
0	[a ; a]$	s3
0 [3	a ; a]$	s2
0 [3 a 2	; a]$	r1
0 [3 S 5	; a]$	r4
0 [3 L 4	; a]$	s7
0 [3 L 4 ; 7	a]$	s2
0 [3 L 4 ; 7 a 2	]$	r1
0 [3 L 4 ; 7 S 8	]$	r3
0 [3 L 4	]$	s6
0 [3 L 4] 6	$	r2
0 S 1	$	OK

ASA Simple LR

- Gramática SLR
  - Uma gramática é SLR sse, para qualquer estado s, as duas condições a seguir são satisfeitas:
    - Para qualquer item A → α.Xβ em s em que X é um terminal, não existe um item completo B → γ. em s com X em Seguidor(B)
    - Para quaisquer dois itens completos A → α. e B →β. em s, Seguidor(A) ∩ Seguidor(B) é vazio.

  - Violações nessas condições geram conflitos, respectivamente:
    - **Carrega-reduz**
      - Resolvido automaticamente pela preferência de carregar ao invés de reduzir
    - **Reduz-reduz**
      - Frequentemente indica erro no projeto da gramática

ASA Simple LR

- Exemplo
  - Conflito carrega-reduz

    <decl> ::= <decl-if> | outra
    <decl-if> ::= if (<exp>) <decl>
    | if (<exp>) <decl> else <decl>
    <exp> ::= 0 | 1

    Na gramática acima, haverá um conflito na transição

    <decl-if> ::= if (<exp>) <decl>**.**
    <decl-if> ::= if (<exp>) <decl>**.** else <decl>

    Trata-se de um conflito carrega-reduz, já que
    - O item completo indica que uma redução deve ocorrer
    - Enquanto o outro item, que o *else* deve ser carregado

ASA Simple LR

- Exemplo
  - Conflito reduz-reduz

    <decl> ::= <ativa-decl> | <atrib-decl>
    <ativa-decl> ::= identificador
    <atrib-decl> ::= <var> := <exp>
    <var> ::= identificador
    <exp> ::= <var> | número

    Na gramática acima, há um estado no qual estão presentes

    <ativa-decl> ::= identificador **.**
    <var> ::= identificador **.**

    ocorre um conflito reduz-reduz, já que
    - Seguidor(<ativa-decl>) = {$} e Seguidor(<var> = {:=, $}
    - Nesse caso, para o símbolo $, a redução deve ocorrer usando 2 regras diferentes: <ativa-decl> ::= identificador e <var> ::= identificador

ASA LR Canônica

- Análise sintática LR Canônica
  - Resolve o conflito reduz-reduz da SLR
  - Item LR(1)
    - Um par composto por um item LR(0) e uma marca de verificação à frente

      [A→ α**.**β,*a*]

      em que A→ α**.**β é um item LR(0) e *a* é uma marca (a verificação à frente)

    - Construção da coleção de itens LR(1) e tabela sintática
      - Essencialmente os mesmos usados na construção da coleção LR(0) e SLR com modificações nas funções fechamento e desvio e considerando-se, agora, LR(1)

ASA LR Canônica

- Exemplo
  - Conflito reduz-reduz

    <decl> ::= <ativa-decl> | <atrib-decl>
    <ativa-decl> ::= identificador
    <atrib-decl> ::= <var> := <exp>
    <var> ::= identificador
    <exp> ::= <var> | número

    Na gramática acima, para o conflito anterior teríamos

    [<ativa-decl> ::= identificador **.**, $]
    [<var> ::= identificador **.**, :=]

    onde o conflito é solucionado já que
    - Os itens LR(1) diferenciam as reduções com base nas suas verificações à frente: reduzir usando a primeira opção se o símbolo à frente for $ e usando a segunda, se for :=

ASA *Look Ahead* LR

- Análise sintática *Look Ahead* LR
  - Características
    - Frequentemente usado na prática
      - Tabelas consideravelmente menores do que as LR canônicas
    - Gramáticas LALR são capazes de expressar
      - A maioria das construções sintáticas comuns das linguagens de programação
    - Não computa o conjunto completo de itens LR(1) na prática
    - Vantagens
      - Preserva alguns dos benefícios da análise sintática SLR
      - Também preserva o menor tamanho do autômato de itens LR(0)

ASA LR

- Análise sintática ascendente LR
  - Vantagens
    - Pode reconhecer praticamente todas as estruturas sintáticas definidas por uma GLC
    - É a mais poderosa e genérica das AS
    - Pode ser implementada eficientemente
    - É capaz de descobrir erros sintáticos o quanto antes
  - Desvantagens
    - Exige manipulação complexa da tabela sintática
    - Construção trabalhosa
      - (manual) de um analisador sintático LR para uma gramática típica de uma linguagem de programação