

# Construção de Compiladores

Prof. Dr. Daniel Lucrédio

DC - Departamento de Computação

UFSCar - Universidade Federal de São Carlos

**Tópico 05 - Análise Sintática LR**

# Referências bibliográficas

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compiladores: Princípios, Técnicas e Ferramentas (2a. edição). Pearson, 2008.

Kenneth C. Loudon. Compiladores: Princípios E Práticas (1a. edição). Cengage Learning, 2004.

Terence Parr. The Definitive Antlr 4 Reference (2a. edição). Pragmatic Bookshelf, 2013.

# Introdução

- Vimos que existem duas formas de se reconhecer uma linguagem através de uma gramática
  - Inferência recursiva
  - Derivação
- Ex: Gramática para expressões aritméticas
  - $V = \{E, I\}$
  - $T = \{+, *, (, ), a, b, 0, 1\}$
  - $P$  = conjunto de regras ao lado
  - $S = E$

$$\begin{array}{lcl} E & \rightarrow & I \\ & | & E + E \\ & | & E * E \\ & | & (E) \\ I & \rightarrow & a \\ & | & b \\ & | & Ia \\ & | & Ib \\ & | & I0 \\ & | & I1 \end{array}$$

# Introdução

- Inferência recursiva

- Dada uma cadeia (conjunto de símbolos terminais)

- Do **corpo** para a **cabeça**

- Ex:  $a^*(a+b00)$

- $a^*(a+b00) \Leftarrow a^*(a+l00) \Leftarrow a^*(a+l0) \Leftarrow a^*(a+l) \Leftarrow a^*(a+E) \Leftarrow a^*(l+E) \Leftarrow a^*(E+E) \Leftarrow a^*(E) \Leftarrow a^*E$   
 $\Leftarrow l^*E \Leftarrow E^*E \Leftarrow E$

- Derivação

- Dada uma cadeia (conjunto de símbolos terminais)

- Da **cabeça** para o **corpo**

- Ex:  $a^*(a+b00)$

- $E \Rightarrow E^*E \Rightarrow l^*E \Rightarrow a^*E \Rightarrow a^*(E) \Rightarrow a^*(E+E) \Rightarrow a^*(l+E) \Rightarrow a^*(a+E) \Rightarrow a^*(a+l) \Rightarrow a^*(a+l0) \Rightarrow$   
 $a^*(a+l00) \Rightarrow a^*(a+b00)$

# Introdução

- Análise sintática descendente
  - Fazer o processo de derivação  
Ou
  - Criar a árvore de análise sintática “de cima para baixo”
    - Análise **top-down**
- Análise sintática ascendente
  - Fazer o processo de inferência  
Ou
  - Criar a árvore de análise sintática “de baixo para cima”
    - Análise **bottom-up**

# Introdução

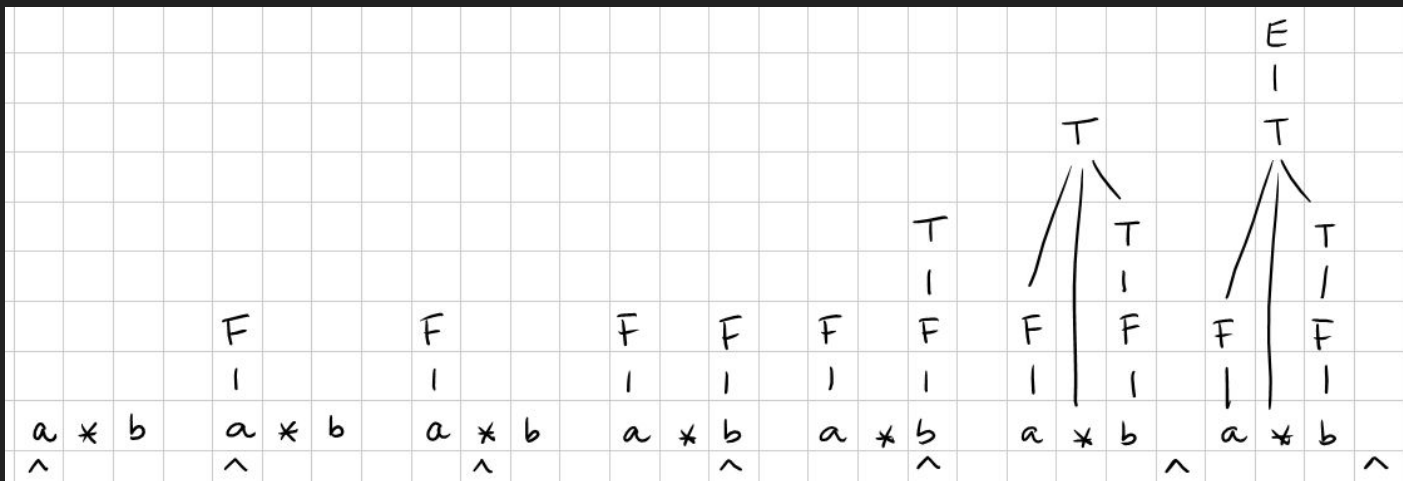
- Exemplo

$$E \rightarrow T + E \mid T$$

$$T \rightarrow F * T \mid F$$

$$F \rightarrow a \mid b \mid (E)$$

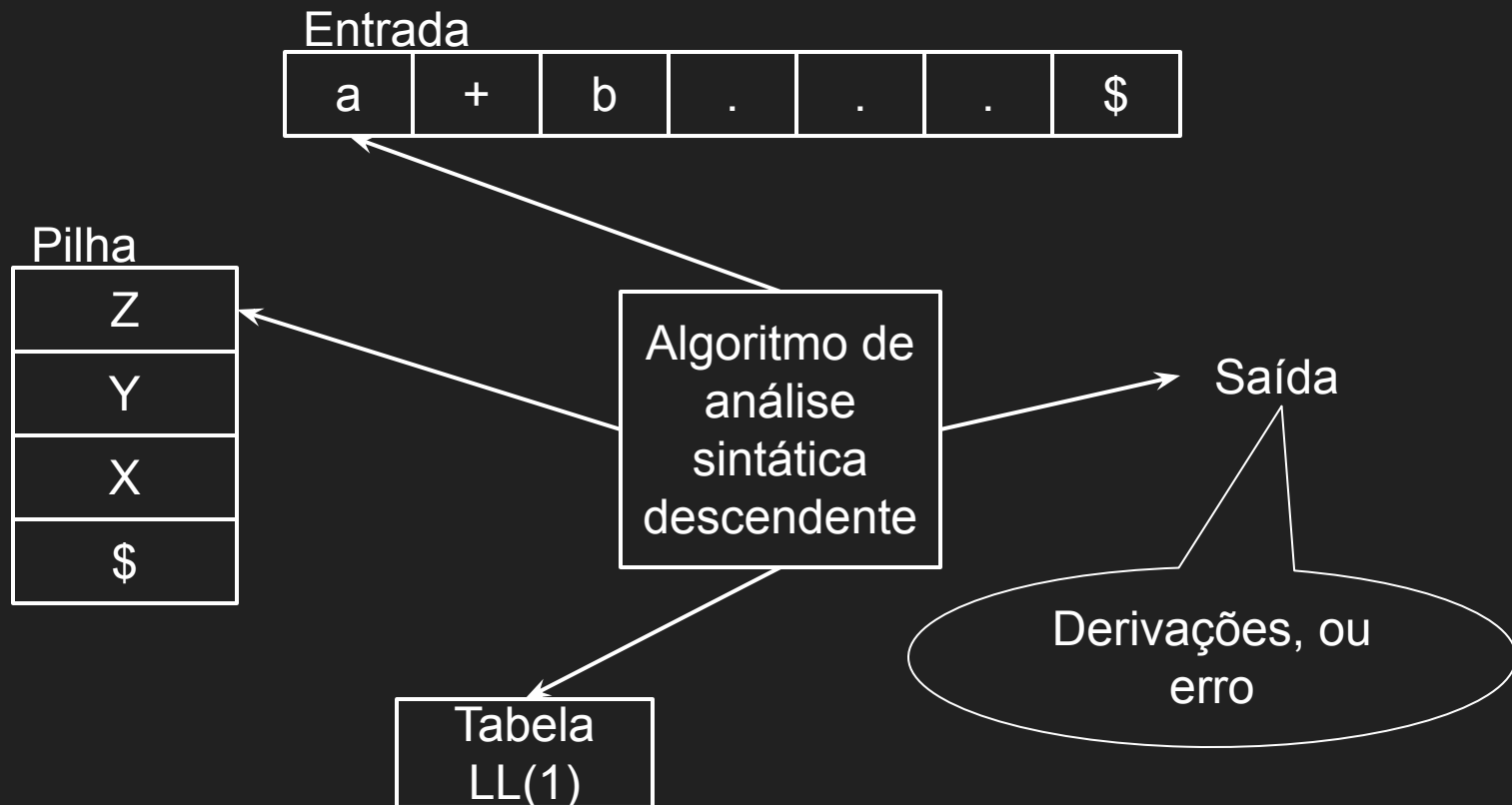
$$\text{Cadeia} = a * b$$



# Introdução

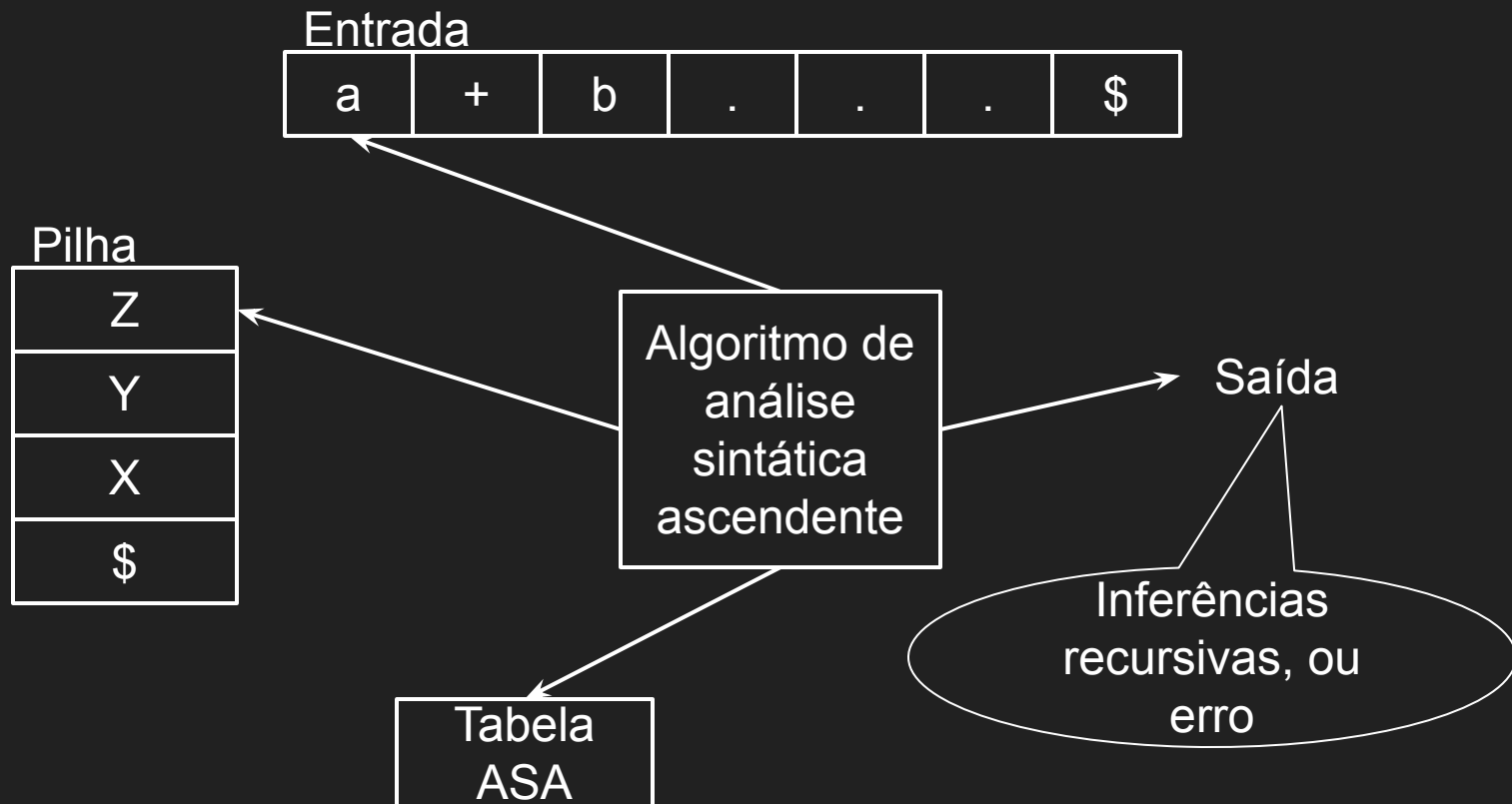
- De acordo com a teoria
  - Analisador sintático = autômato de pilha
  - Portanto, para ambos os casos, um autômato de pilha deve ser suficiente
- Na análise sintática descendente, já vimos como fazer
  - A pilha armazena os símbolos a serem substituídos
  - Quando a pilha esvaziar, acabou
- Na análise sintática ascendente
  - A pilha vai armazenar os símbolos aguardando “redução”
  - Quando sobrar só o símbolo inicial na pilha, acabou

# Introdução





# Introdução



# Introdução

- Análise sintática descendente = método (algoritmo) que produz uma derivação mais à esquerda para uma cadeia da entrada
- O problema principal em cada passo é determinar qual produção aplicar
- Sendo que os tokens são lidos da esquerda para a direita
- Ex:

- Entrada:  $a + b * c$
- Token atual =  $a$
- Símbolo inicial:  $E$
- Possíveis produções de  $E$ :
  - $E \rightarrow E + E$
  - $E \rightarrow E * E$
  - $E \rightarrow (E)$

Qual  
escolher?

$E$	$\rightarrow$	$I$
	$ $	$E + E$
	$ $	$E * E$
	$ $	$(E)$
$I$	$\rightarrow$	$a \mid b \mid c$

# Exemplo: análise sintática descendente

- Gramática:  $S \rightarrow n + S \mid n$
- Cadeia:  $n + n$

Escolha é  
guiada pela  
tabela LL

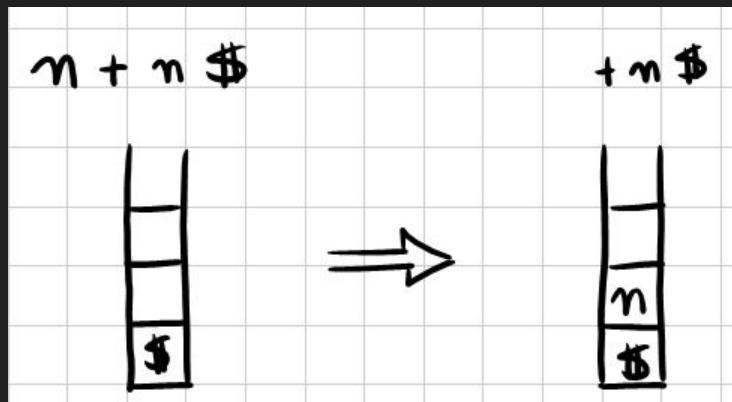
Casamento	Pilha	Entrada	Ação
	<u>S</u> \$	<u>n</u> +n\$	$S \rightarrow n+S$
<u>n</u>	<u>n</u> +S\$	<u>n</u> +n\$	match
<u>n</u> <u>+</u>	<u>+</u> S\$	<u>+</u> n\$	match
	<u>S</u> \$	<u>n</u> \$	$S \rightarrow n$
<u>n</u> + <u>n</u>	<u>n</u> \$	<u>n</u> \$	match
	<u>\$</u>	<u>\$</u>	aceita

# Introdução

- Na análise sintática ascendente, temos um processo diferente
- Para reconhecer uma cadeia de entrada:
  - Empilha
    - Os símbolos da cadeia de entrada
  - Reduz
    - O lado direito de uma produção no topo da pilha, substituindo-o pelo lado esquerdo da produção
- Os passos 1 e 2 são repetidos até que
  - ACEITA – os símbolos da cadeia de entrada foram consumidos e a pilha possui apenas o símbolo inicial da gramática
  - OU
  - ERRO – o processo foi interrompido antes de chegar ao final

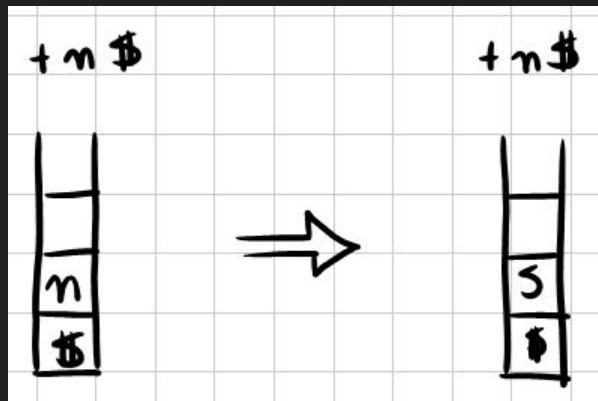
# Análise sintática ascendente

- Empilhamento
  - Consiste em remover um símbolo da entrada e adicioná-lo ao topo da pilha
- Ex:
  - Gramática =  $S \rightarrow S + n \mid n$
  - Entrada =  $n + n$



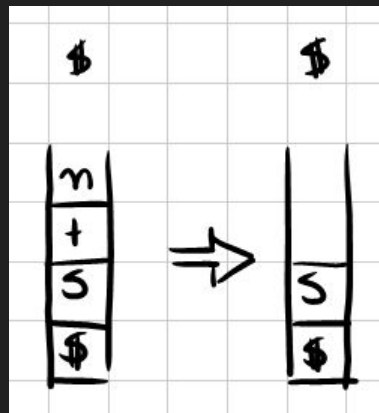
# Análise sintática ascendente

- Redução
  - Consiste em substituir símbolos no topo da pilha por um único símbolo
    - Não consome a entrada
- Ex:
  - Gramática =  $S \rightarrow S + n \mid n$
  - Entrada =  $n + n$



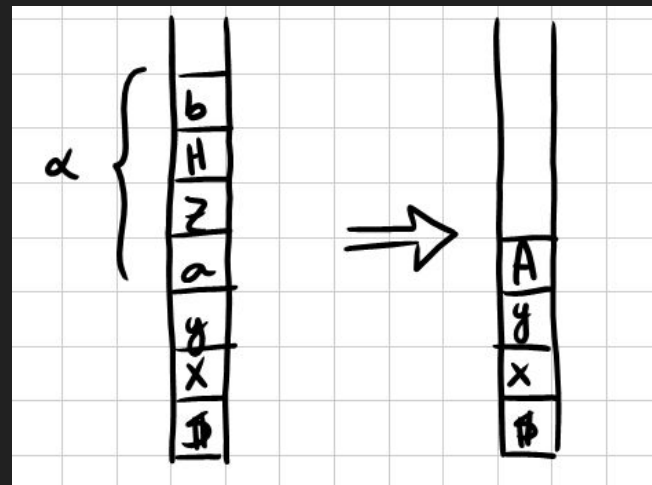
# Análise sintática ascendente

- Redução
  - Consiste em substituir símbolos no topo da pilha por um único símbolo
    - Não consome a entrada
- Ex:
  - Gramática =  $S \rightarrow S + n \mid n$
  - Entrada =  $n + n$



# Análise sintática ascendente

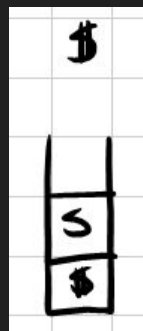
- Quando empilhar/reduzir?
- Conceito de “gancho” (handle)
  - Para cada produção  $A \rightarrow \alpha$
  - $\alpha$  é um “gancho”
    - Quando  $\alpha$  aparecer no topo da pilha, posso substituir por A
- Ex:  $A \rightarrow aZHb$ 
  - $\alpha = aZHb$





# Análise sintática ascendente

- Aceita
  - Quando consumir toda a entrada
  - Pilha contém somente o símbolo inicial
- Ex:
  - Gramática =  $S \rightarrow S + n \mid n$
  - Entrada =  $n + n$



# Análise sintática ascendente

- Exemplo

- Gramática:  $S \rightarrow S + n \mid n$
- Entrada:  $n + n$

Agora escrevemos a pilha da esquerda para a direita, para facilitar

Apareceu  
um  
“gancho”  
aqui

E aqui  
também

Pilha	Entrada	Ação
\$	<u>n</u> +n\$	empilha n
\$ <u>n</u>	<u>+</u> n\$	reduz $S \rightarrow n$
\$S	<u>+</u> n\$	empilha +
\$S+	<u>n</u> \$	empilha n
\$ <u>S+n</u>	<u>\$</u>	reduz $S \rightarrow S+n$
\$S	<u>\$</u>	aceita

# Análise sintática ascendente

- Desafio
  - Detectar o aparecimento do “gancho” na pilha
  - Exige olhar um ou mais símbolos da pilha
  - E também olhar símbolos à frente na entrada
    - Normalmente, busca-se olhar somente um símbolo à frente, por uma questão de eficiência

# Análise sintática ascendente LR

# ASA LR

- Analisador LR (k)
  - Left to right with Rightmost derivation
  - Lê a sentença em análise da esquerda para a direita
  - Produz uma derivação mais à direita ao reverso
    - Inferência recursiva
  - Considerando-se k símbolos na cadeia de entrada

# Tabela de análise LR

- Existem diferentes tipos de tabelas LR
  - Cada uma com vantagens/desvantagens (veremos depois)
- A tabela LR é dividida em duas
  - Ação
  - Transição
- A tabela é construída diretamente a partir da gramática
- Estados = armazenam a situação atual de leitura
  - Permitem detectar o aparecimento de um “gancho”

# Exemplo de tabela de análise LR

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

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			

# Tabela de análise LR

- Os códigos para as ações são:
  - $si = \textit{shift } i$ 
    - “avança na entrada e empilha o estado  $i$  na pilha”
  - $rj = \textit{reduce } j$ 
    - “reduz segundo a produção de número  $j$ ”
  - OK
    - “aceita a entrada”
  - Entrada em branco
    - Erro sintático



# Algoritmo de análise LR

- ENTRADA: uma cadeia de entrada  $w$  e uma tabela de análise LR com as ações e transições definidas para uma gramática  $G$
- SAÍDA: se  $w$  está em  $L(G)$ , os passos de inferência recursiva (análise ascendente para  $w$ ). Caso contrário, uma indicação de erro
- CONDIÇÕES INICIAIS:
  - $w\$$  no buffer de entrada
  - $s_0$  na pilha (estado inicial)

# Algoritmo de análise LR

```
a := primeiro símbolo de w$;
while(1) { /* repita indefinidamente */
    s := estado no topo da pilha;
    if(ACAO[s,a] = "shift t") {
        empilha t;
        a := próximo símbolo da entrada;
    } else if (ACAO[s,a] = "reduce A -->  $\beta$ ") {
        desempilha  $|\beta|$  símbolos;
        t := topo da pilha;
        empilha TRANSICAO[t,A];
        imprima "A -->  $\beta$ "
    } else if (ACAO[s,a] = "OK") pare; /* fim */
    else erro;
}
```

- # Exemplo

Entrada = **id \* id + id**

[illegible]

- 1. E → E + T
- 2. E → T
- 3. T → T \* F
- 4. T → F
- 5. F → (E)
- 6. F → id

# Exemplo

Entrada = **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	Símbolos	Entrada	Ação
0		id*id+id\$	s5
0 5	id	*id+id\$	r6
0 3	F	*id+id\$	r4
0 2	T	*id+id\$	s7
0 2 7	T *	id+id\$	s5
0 2 7 5	T * id	+id\$	r6
0 2 7 10	T * F	+id\$	r3
0 2	T	+id\$	r2
0 1	E	+id\$	s6
0 1 6	E +	id\$	s5
0 1 6 5	E + id	\$	r6
0 1 6 3	E + F	\$	r4
0 1 6 9	E + T	\$	r1
0 1	E	\$	OK

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

# Exemplo

Entrada = **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			

[illegible]

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

# Exemplo

Entrada = **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	Símbolos	Entrada	Ação
0		id*(id+id)\$	s5
0 5	id	*(id+id)\$	r6
0 3	F	*(id+id)\$	r4
0 2	T	*(id+id)\$	s7
0 2 7	T *	(id+id)\$	s4
0 2 7 4	T * (	id+id)\$	s5
0 2 7 4 5	T * ( id	+id)\$	r6
0 2 7 4 3	T * ( F	+id)\$	r4
0 2 7 4 2	T * ( T	+id)\$	r2
0 2 7 4 8	T * ( E	+id)\$	s6
0 2 7 4 8 6	T * ( E +	id)\$	s5
0 2 7 4 8 6 5	T * ( E + id	)\$	r6
0 2 7 4 8 6 3	T * ( E + F	)\$	r4
0 2 7 4 8 6 9	T * ( E + T	)\$	r1
0 2 7 4 8	T * ( E	)\$	s11
0 2 7 4 8 11	T * ( E )	\$	r5
0 2 7 10	T * F	\$	r3
0 2	T	\$	r2
0 1	E	\$	OK

- # Exemplo

Entrada = **id** \* (id

[illegible]

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

# Exemplo

Entrada = **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	Símbolos	Entrada	Ação
0		id*(id\$	s5
0 5	id	*(id\$	r6
0 3	F	*(id\$	r4
0 2	T	*(id\$	s7
0 2 7	T *	(id\$	s4
0 2 7 4	T * (	id\$	s5
0 2 7 4 5	T * ( id	\$	r6
0 2 7 4 3	T * ( F	\$	r4
0 2 7 4 2	T * ( T	\$	r2
0 2 7 4 8	T * ( E	\$	erro



# Conflitos

- Conflito empilha-reduz

$\text{decl} \rightarrow \text{decl-if} \mid \text{'outra'}$

$\text{decl-if} \rightarrow \text{'if' '(' exp ')'} \text{ decl}$

$\quad \mid \text{'if' '(' exp ')'} \text{ decl 'else' decl}$

$\text{exp} \rightarrow \text{'0'} \mid \text{'1'}$

- Na gramática acima, haverá um conflito na transição a partir das seguintes possibilidades para o mesmo gancho (obs: o ponto **.** indica o momento atual da leitura):

$[\text{decl-if} \rightarrow \text{'if' '(' exp ')'} \text{ decl} \mathbf{.}]$

$[\text{decl-if} \rightarrow \text{'if' '(' exp ')'} \text{ decl} \mathbf{.} \text{'else' decl}]$

- Trata-se de um conflito empilha-reduz, já que
  - A primeira possibilidade indica que deve haver uma redução
  - Enquanto a segunda indica que o 'else' deve ser empilhado

# Conflitos

- Conflito reduz-reduz

`decl → ativa-decl | atrib-decl`

`ativa-decl → ID`


`atrib-decl → var ':= ' exp`

`var → ID`

`exp → var | NUM`

- Na gramática acima, há uma situação com duas possibilidades para um mesmo gancho

`[ativa-decl→ID`  `]`

`[var→ID`  `]`

- Ocorre um conflito reduz-reduz, já que, no aparecimento do gancho “ID” na pilha, há duas possíveis reduções:
  - `ativa-decl→ID`
  - `var→ID`

# Conflitos

- Resolução de conflitos

- Empilha-reduz

- Opção default (**sempre empilhar**)
    - Resolve a ambiguidade do “else sobrando”**, pois um else deve sempre ser agrupado com o if mais próximo

- Ex:

if (1) then if (2) then outra else outra

Decisão **errada**:

Pilha	Entrada	Ação
if (exp) then if(exp) then outra	else outra \$	reduce
if (exp) then decl-if	else outra \$	shift
if (exp) then decl-if else	outra \$	shift
if (exp) then decl-if else outra	\$	reduce
decl-if	\$	

# Conflitos

- Resolução de conflitos

- Empilha-reduz

- Opção default (**sempre empilhar**)
    - Resolve a ambiguidade do “else sobrando”**, pois um else deve sempre ser agrupado com o if mais próximo

- Ex:

if (1) then if (2) then outra else outra

Decisão **certa**:

Pilha	Entrada	Ação
if (exp) then if(exp) then outra	else outra \$	shift
if (exp) then if(exp) then outra else	outra \$	shift
if (exp) then if(exp) then outra else outra	\$	reduce
if (exp) then decl-if	\$	reduce
decl-if	\$	

# Conflitos

- Resolução de conflitos
  - Empilha-reduz
    - Associatividade/precedência
    - Resolve ambiguidades de expressões
- Consiste em definir explicitamente (fora da gramática) a relação de precedência/associatividade entre os terminais
  - Ex:  $* > +$ ,  $+ > +$ , etc
- No momento da dúvida:
  - $a$  = o terminal mais à direita na pilha
  - $b$  = a entrada atual
  - Se  $a > b$ , reduz
  - Se  $a < b$ , empilha

# Conflitos

• Ex:

id+id\*id

Pilha	Entrada	Ação
E + E	* id \$	shift
E + E *	id \$	shift
E + E * id	\$	reduce
E + E * E	\$	reduce
E + E	\$	reduce
E	\$	...

Momento de dúvida, pois existe um estado com as seguintes opções:

$[E \rightarrow E + E \cdot]$  reduz?

$[E \rightarrow E \cdot * E]$  ou empilha?

Neste exemplo

a = +

b = \*

+ < \*, portanto empilha

Regras de associação/precedência:

\* > +, + > +

# Conflitos

• Ex:

id+id+id

Pilha	Entrada	Ação
E + E	+ id \$	reduce
E	+ id \$	shift
E +	id \$	shift
E + id	\$	reduce
E + E	\$	reduce
E	\$	...

Momento de dúvida, pois existe um estado com os seguintes itens:

[ E → E + E . ]

[ E → E . + E ]

e + está em seguidores(E)

Neste exemplo

a = +

b = +

+ > +, portanto reduz

Regras de associação/precedência:

\* > +, + > +

# Conflitos

- Normalmente indicam ambiguidade ou outro problema no projeto da gramática
- Precisa re-escrever a gramática
  - Não existe regra para isso
  - Ou alterar diretamente a tabela LR



# Tipos de tabelas LR - complexidade na construção

- SLR(1): Simple LR
  - Simples de construir
- LR(1) canônica
  - Média complexidade para construir
- LALR(1): LookAhead LR
  - Alta complexidade para construir
- Porém existem algoritmos automatizados, portanto essa complexidade não é um fator preponderante

# Tipos de tabelas LR - conflitos

- ~~SLR(1): Simple LR~~
  - Muitos conflitos
- LR(1) canônica
  - Menos conflitos que SLR
- LALR(1): LookAhead LR
  - Menos conflitos que SLR
- Pelo quesito de conflitos, SLR pode ser descartada

# Tipos de tabelas LR - eficiência

- ~~SLR(1): Simple LR~~

- Tamanho pequeno

- ~~LR(1) canônica~~

- Tamanho grande / pouca eficiência

- LALR(1): LookAhead LR

- Tamanho pequeno

- Pelo quesito de eficiência, resta a LALR como **a mais indicada**

Fim