

Estudos sobre Compiladores

Análise Sintática Top-Down

- **Autoria:**

- Agnelo Rocha da Silva
- Pesquisas realizadas:
 - Notas de aula do Prof. Riverson Rios (UFC)
 - Livro "Compiladores - Princípios, Técnicas e Ferramentas (A.Aho, R.Sethi, J.Ullman) - Cap.4

- **Contexto Acadêmico:**

- Curso de Ciências da Computação (UFC)
- Disciplina de Compiladores (2007-1)
- Prof.: Riverson Rios

- **Contexto na Disciplina:**

- Análise Sintática
 - Analisador Sintático e Gramáticas
 - Introdução de Conceitos
 - Analisador Sintático Descendente Recursivo (com Retrocesso)
 - Analisador Sintático LL(1)
-

A sintaxe das construções de uma linguagem de programação pode ser descrita pelas gramáticas livres de contexto (GLC).

Uma gramática oferece, para uma linguagem de programação, uma especificação sintática precisa e fácil de entender.

Para certas classes de gramáticas, podemos construir automaticamente um analisador sintático que determine se o programa-fonte está sintaticamente bem formado.

Note que nem todas gramáticas GLC poderão ter um analisador sintático eficiente correspondente.

O processo de construção do analisador sintático pode revelar ambigüidades sintáticas na GLC proposta.

Os compiladores mais eficientes utilizarão analisadores sintáticos de dois grupos:

1. Top-down

- * constroem árvores sintáticas do topo (raiz) para o fundo (folhas) - veremos um exemplo a seguir;
- * a entrada é varrida da esquerda para a direita, um símbolo por vez;
- * exemplo: analisador sintático preditivo para a gramática $\underline{L}(1)$; note que o L sublinhado significa Left-to-right.

2. Bottom-up

- * constroem árvores sintáticas das folhas até a raiz - veremos um exemplo a seguir;
- * a entrada também é varrida da esquerda para a direita, um símbolo por vez;
- * exemplo: analisador sintático $\underline{L}R(1)$; note que o L sublinhado significa Left-to-right.

Comparação Top-down vs. Bottom-up

Gramática:

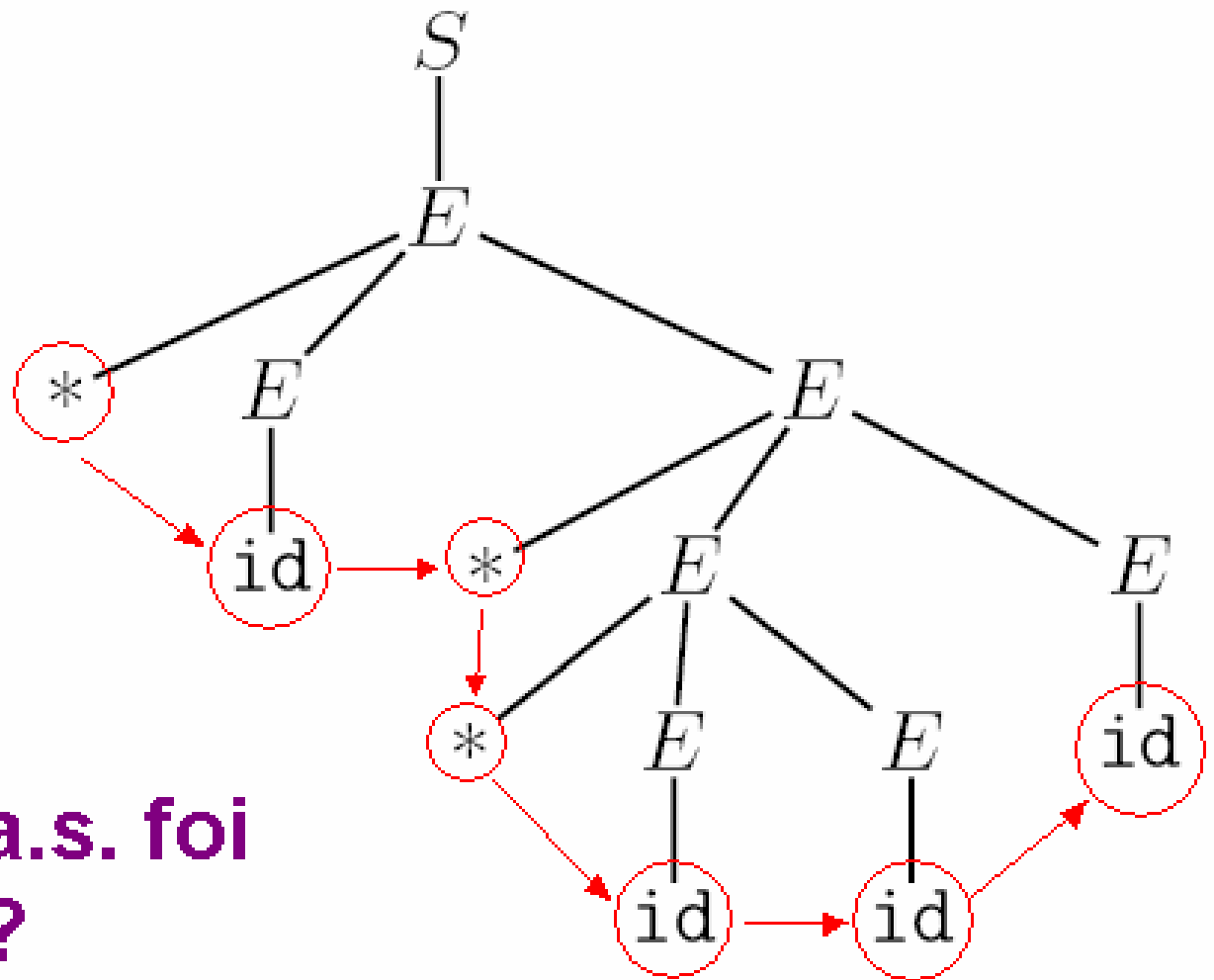
$$S \rightarrow E$$
$$E \rightarrow id$$
$$E \rightarrow * E E$$

Entrada:

$$*id \quad * \quad *id \quad id \quad id$$

Como esta a.s. foi construída ?

Árvore Sintática:



Comparação Top-down vs. Bottom-up

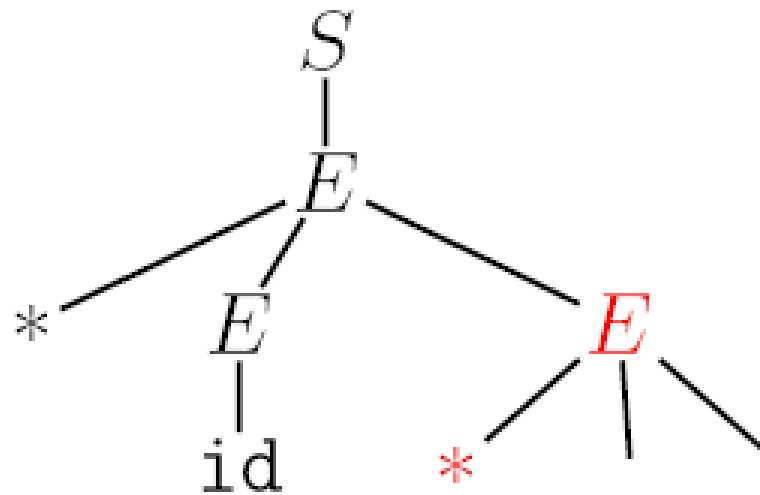
Gramática:

$$S \rightarrow E$$
$$E \rightarrow id$$
$$E \rightarrow * E E$$

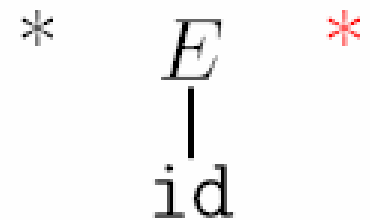
Entrada:

* id * * id id id
↑ ↑ ↑

LL Parse



LR Parse



Sem entrar em detalhes, podemos observar uma diferença fundamental, entre as classes de analisadores sintáticos, quanto à forma de construírem suas árvores sintáticas.

Algumas linguagens não podem ser geradas por nenhuma gramática. Um exemplo disso são as linguagens que não são livre de contexto.

Esses casos, quando existem nas linguagens de programação, constituem uma exceção e são tratados, em geral, pela análise semântica.

Mesmo considerando apenas as linguagens livres de contexto, somente algumas sub-classes de GLCs são geradas pelos analisadores sintáticos top-down e bottom-up.

Felizmente, várias dessas subclasses de gramáticas, como a LL e LR, são suficientemente expressivas para descrever a maioria das construções sintáticas em linguagens de programação.

Um analisador sintático pode ser criado especificamente para uma gramática ou um conjunto de gramáticas LL conhecidas. Neste caso, é comum implementarmos tal analisador sintático "manualmente".

Porém, os analisadores da classe mais ampla das gramáticas LR são usualmente construídos através de ferramentas automatizadas, como o Bison.

A entrada de um analisador sintático é um fluxo de tokens produzido pelo analisador léxico (ex.: *id**id).

A saída do analisador sintático é alguma representação da árvore sintática que representa a entrada.

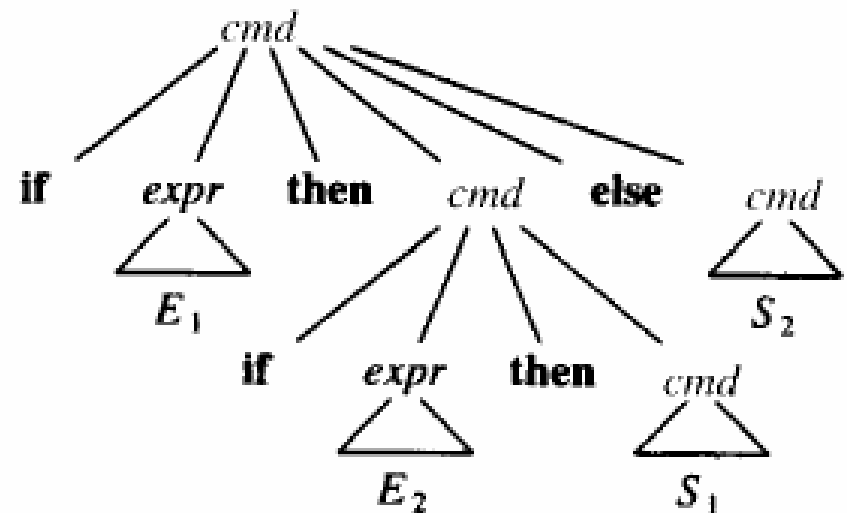
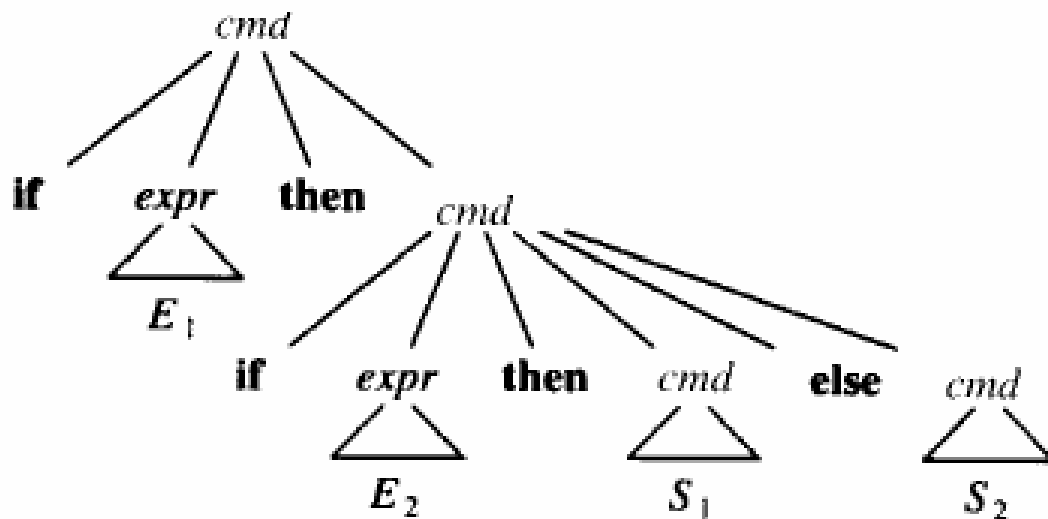
Problemas associados a uma gramática

Ambigüidade

Exemplo: $C \Rightarrow$ if E then C
 | if E then C else C

entrada: if E1 then if E2 then S1 else S2

árvores sintáticas produzidas:



Problemas associados a uma gramática

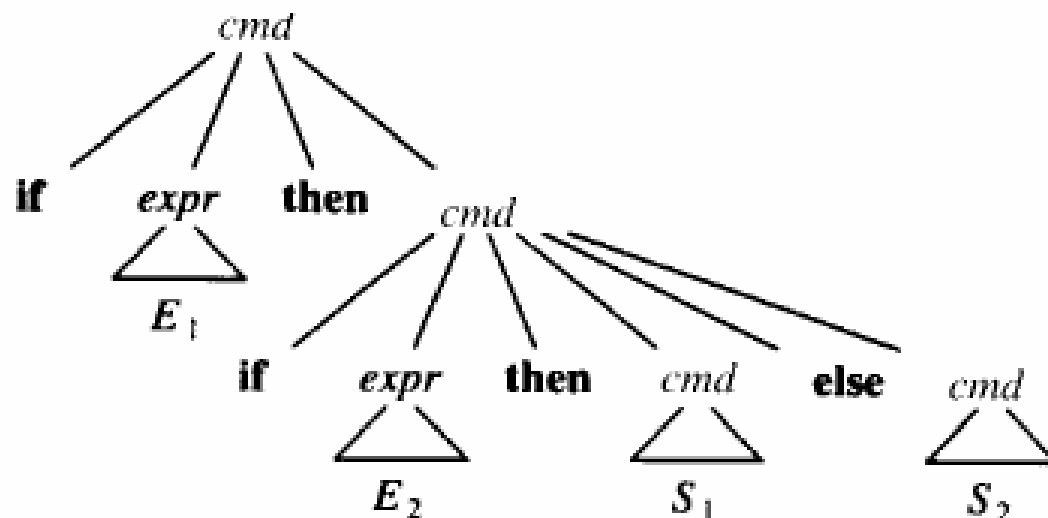
Ambigüidade

Solução: reescrever a gramática

Exemplo:
C $\Rightarrow$ A
| N
A $\Rightarrow$ if E then A else A
N $\Rightarrow$ if E then C
| if E then A else N

entrada: if E1 then if E2 then S1 else S2

árvore sintática produzida:



Problemas associados a uma gramática

Recursividade à esquerda

Problema: analisadores sintáticos top-down não processam

Solução: modificar a gramática criando NT(s) extra(s)

Exemplo:

$$\begin{array}{l} E \Rightarrow \textcircled{E} + T \\ \quad | \quad T \\ T \Rightarrow \textcircled{T} * F \\ \quad | \quad F \\ F \Rightarrow (E) \\ \quad | \quad \text{id} \end{array}$$



$$\begin{array}{l} E \Rightarrow TE' \\ E' \Rightarrow +TE' \\ \quad | \quad \epsilon \\ T \Rightarrow FT' \\ T' \Rightarrow *FT' \\ \quad | \quad \epsilon \\ F \Rightarrow (E) \\ \quad | \quad \text{id} \end{array}$$

Problemas associados a uma gramática

Indecisão quanto à regra a ser utilizada

Problema: analisadores sintáticos preditivos (como o LL(1))
ficam com conflitos

Solução: modificar a gramática com fatoração à esquerda

Exemplo:

$$\begin{array}{c} A \Rightarrow \alpha \beta_1 \\ | \\ \alpha \beta_2 \end{array} \quad \longrightarrow \quad \begin{array}{c} A \Rightarrow \alpha A' \\ A' \Rightarrow \beta_1 \\ | \\ \beta_2 \end{array}$$

Um analisador sintático top-down pode ser do tipo:

1. analisador sintático de descendência recursiva (pode usar retrocesso)
2. analisador sintático preditivo (não usa retrocesso)

Ambos analisadores tentam sempre uma derivação mais à esquerda (Leftmost), ou seja a "procura" se dá pelo "casamento" do que está no lado esquerdo das regras. Como a primeira regra inclui o símbolo inicial, que é a raiz da árvore sintática, a construção da mesma se dará sempre de cima para baixo.

Descendência Recursiva com Retrocesso

Aplica as derivações:

- * de cima para baixo (começa pelo símbolo NT inicial);
- * casa (matching) a extremidade esquerda (leftmost) das regras:

$$\begin{array}{c} A \\ \uparrow \end{array} \Rightarrow \alpha \beta$$

- * analisa um caracter por vez, da esquerda para a direita da cadeia de entrada (todos os analisadores estudados nesta disciplina de Compiladores fazem assim também):

$\begin{array}{ccccccc} * & id & * & * & id & id & id \\ \uparrow & \uparrow & \uparrow & & & & \end{array}$

Descendência Recursiva com Retrocesso

Aplica as derivações: (cont.)

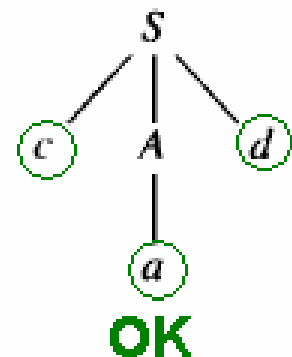
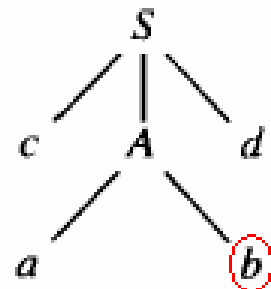
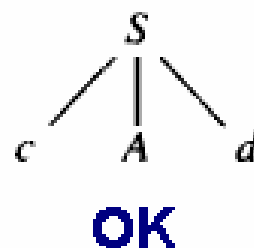
- * quando existe mais de uma possível regra a ser aplicada, tenta usar uma delas e, por fim, caso a cadeia inicial não seja aceita, faz um retrocesso e tenta a(s) outra(s) alternativa(s):

Gramática:

$S \Rightarrow cAd$

$A \Rightarrow ab$
|
a

Cadeia inicial: cad



ERRO
(retroceder)

Descendência Recursiva Preditiva LL(1)

Aplica as derivações:

- * de cima para baixo (começa pelo símbolo NT inicial);
- * casa (matching) a extremidade esquerda (leftmost) das regras:

$$\begin{array}{c} A \\ \uparrow \end{array} \Rightarrow \alpha \beta$$

- * analisa um caracter por vez, da esquerda para a direita da cadeia de entrada (todos os analisadores estudados nesta disciplina de Compiladores fazem assim também):

* id * id id id
↑ ↑ ↑

idem ao anterior !!

Descendência Recursiva Preditiva LL(1)

Aplica as derivações: (cont.)

- * Utiliza uma tabela sintática preditiva (*look ahead* de 1 símbolo).
- * Não faz recursão e não faz retrocessos: todo o processo é determinístico (e eficiente!!) graças ao uso da tabela preditiva. Isso só vale se a gramática for LL(1). Veremos depois o que isso significa.
- * Para construir a tabela, precisa dos conjuntos First e Follow dos NTs da gramática.

Descendência Recursiva Preditiva LL(1)

Construção da Tabela Preditiva LL(1):

Gramática:

$S \Rightarrow cAd$

$A \Rightarrow \begin{matrix} ab \\ | \\ a \end{matrix}$

—— Símbolos Terminais ——

Símbolos
NTs

	a	b	c	d	\$
S					
A					

Descendência Recursiva Preditiva LL(1)

Algoritmo de Construção da Tabela Preditiva LL(1)

Seja a regra de derivação $A \rightarrow \alpha$ em G :

1. Se $a \in \text{First}_\alpha$, então:
posição $[A, a] = A \rightarrow \alpha$
2. Se $\epsilon \in \text{First}_\alpha$ e $b \in \text{Follow}_A$, então:
posição $[A, b] = A \rightarrow \alpha$
3. Se $\epsilon \in \text{First}_\alpha$ e $\$ \in \text{Follow}_A$, então:
posição $[A, \$] = A \rightarrow \alpha$

Notação $\left\{ \begin{array}{l} G \text{ é uma gramática} \\ a, b \text{ são símbolos } T \text{ (terminais)} \\ A \text{ é símbolo } NT \text{ (não terminal)} \\ \alpha \text{ é uma derivação qualquer (pode ser um símbolo } T, NT \text{ ou combinação)} \end{array} \right.$

Obs.: repetir o algoritmo para toda regra de derivação em G

Descendência Recursiva Preditiva LL(1)

Construção da Tabela Preditiva LL(1): (cont.)

Gramática:

$S \Rightarrow cAd$

$A \Rightarrow ab$
|
a

	First	Follow
S	c	\$
A	a	d

—— Símbolos Terminais ——

	a	b	c	d	\$
S			$S \Rightarrow cAd$		
A	$A \Rightarrow ab$ $A \Rightarrow a$				

O que isso significa ?

Descendência Recursiva Preditiva LL(1)

Um erro muito comum, aprendido do exemplo anterior, é acharmos que o analisador LL(1) ao utilizar uma tabela preditiva estaria "imune" ao problema de retrocesso que ocorre com o analisador de descendência recursiva

Mesmo com algumas medidas - eliminando-se a recursividade à esquerda e aplicando-se fatoração à esquerda sempre que "indecisões" de regras possam surgir - ainda é possível ocorrer da gramática possuir entradas multiplamente definidas!!

A conclusão: essa gramática não é LL(1).

Ou muda-se a gramática (será que dá para "adaptá-la"?) ou implementa-se o analisador LL(1) com tratamento de retrocesso (e a eficiência que tínhamos antes?).

Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$
| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$
| ϵ

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

Tabela Sintática Preditiva LL(1)

	a	,	(	)	\$
S					
L					
L'					

Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

$S \rightarrow (L) \equiv A \rightarrow \alpha$

Regra 1 do algoritmo :

$\text{First}_\alpha = \{ (\}$

$\text{pos.}[S, (] = S \rightarrow (L)$

Tabela Sintática Preditiva LL(1)

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

	a	,	(	)	\$
S			$S \rightarrow (L)$		
L					
L'					

Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

$S \rightarrow a$

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

$\mid \epsilon$

$S \rightarrow a \equiv A \rightarrow \alpha$

Regra 1 do algoritmo :

$\text{First}_\alpha = \{ a \}$

$\text{pos.}[S, a] = S \rightarrow a$

Tabela Sintática Preditiva LL(1)

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

	a	,	(	)	\$
S	$S \rightarrow a$		$S \rightarrow (L)$		
L					
L'					

Exemplo de uso do analisador LL(1)

$$\begin{array}{l} S \rightarrow (L) \\ \quad | \quad a \\ L \rightarrow SL' \\ L' \rightarrow ,SL' \\ \quad | \quad \epsilon \end{array}$$

$L \rightarrow SL' \equiv A \rightarrow \alpha$

Regra 1 do algoritmo :

$\text{First}_\alpha = \{ a, (\}$

$\text{pos.}[L, a] = L \rightarrow SL'$

$\text{pos.}[L, (] = L \rightarrow SL'$

Tabela Sintática Preditiva LL(1)

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

	a	,	(	)	\$
S	$S \rightarrow a$		$S \rightarrow (L)$		
L	$L \rightarrow SL'$		$L \rightarrow SL'$		
L'					

Exemplo de uso do analisador LL(1)

$$\begin{array}{l} S \rightarrow (L) \\ \quad | \quad a \\ L \rightarrow SL' \\ L' \rightarrow ,SL' \\ \quad | \quad \epsilon \end{array}$$

$L' \rightarrow ,SL' \equiv A \rightarrow \alpha$

Regra 1 do algoritmo :

$\text{First}_\alpha = \{ , \}$

$\text{pos.}[L', ,] = L' \rightarrow ,SL'$

Tabela Sintática Preditiva LL(1)

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

	a	,	(	)	\$
S	$S \rightarrow a$		$S \rightarrow (L)$		
L	$L \rightarrow SL'$		$L \rightarrow SL'$		
L'		$L' \rightarrow ,SL'$			

Exemplo de uso do analisador LL(1)

$$\begin{array}{c} S \rightarrow (L) \\ | \quad a \end{array}$$

$$L \rightarrow SL'$$

$$L' \rightarrow , SL'$$

$$L' \rightarrow \epsilon$$

$$L' \rightarrow \epsilon \equiv A \rightarrow \alpha$$

Regra 2 do algoritmo :

$$\text{First}_\alpha = \{ \epsilon \} \text{ e } \text{Follow}_A = \{) \}$$

$$\text{pos.}[L',)] = L' \rightarrow \epsilon$$

Tabela Sintática Preditiva LL(1)

	First	Follow
S	a (	\$,)
L	a (	)
L'	, ϵ	)

	a	,	(	)	\$
S	$S \rightarrow a$		$S \rightarrow (L)$		
L	$L \rightarrow SL'$		$L \rightarrow SL'$		
L'		$L' \rightarrow , SL'$		$L' \rightarrow \epsilon$	

Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Consultar pos.[S, (] na tabela:

	a	,	(
S	$S \rightarrow a$		$S \rightarrow (L)$

Pela regra acima, substituir S por (L) na pilha:

Buffer de entrada:

(a) \$



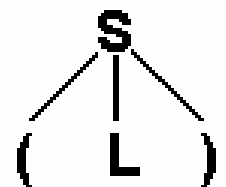
Pilha:

S
\$

~~\$~~
\$

(
L
)
\$

Árvore sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow \epsilon, SL'$

| ϵ

Teste de aceitação de cadeia:

Casamento OK: topo de pilha e ponteiro de buffer de entrada com mesmo símbolo.

Eliminar (da pilha e andar com ponteiro do buffer

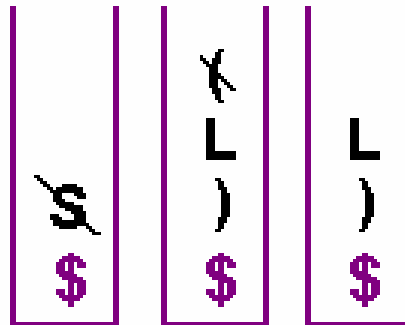
Buffer de entrada:

(a) \$

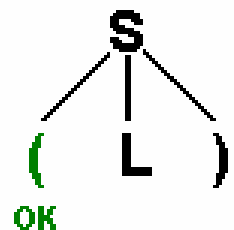


Pilha:

(
L
)
\$



Árvore sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Consultar pos.[L, a] na tabela:

	a
S	$S \rightarrow a$
L	$L \rightarrow SL'$

Pela regra acima, substituir L por SL' na pilha:

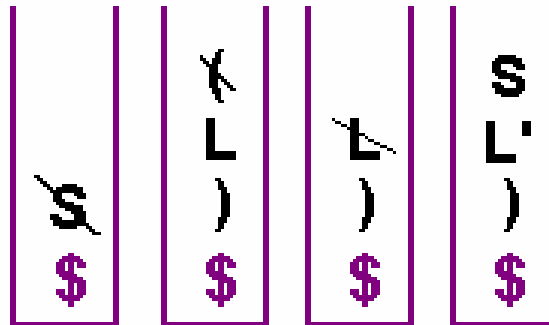
Buffer de entrada:

(a) \$

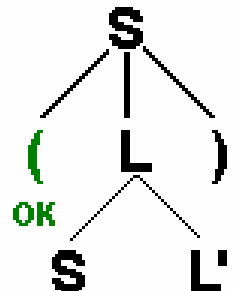


Pilha:

L
)
\$



Árvore sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Consultar pos.[S, a] na tabela:

	a
S	$S \rightarrow a$

Pela regra acima, substituir S por a na pilha:

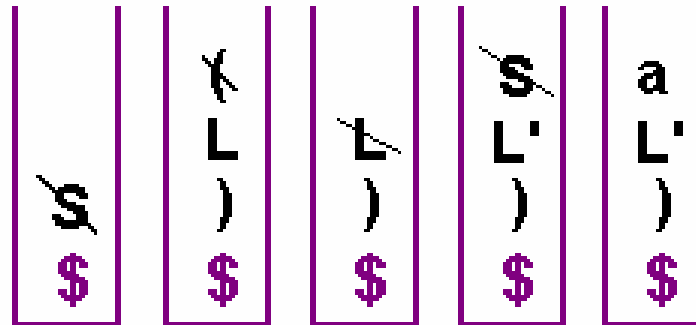
Buffer de
entrada:

(a) \$

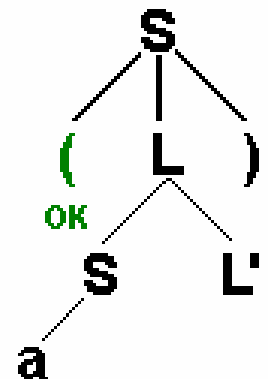


Pilha:

S
L'
)
\$



Árvore
sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Casamento OK: topo de pilha e ponteiro de buffer de entrada com mesmo símbolo.

Eliminar **a** da pilha e andar com ponteiro do buffer :

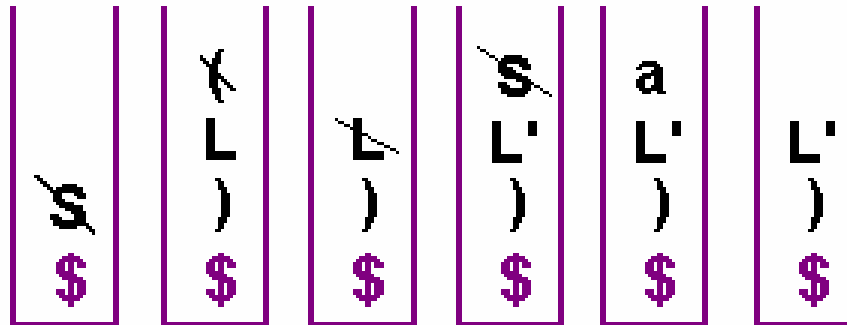
Buffer de entrada:

(a) \$

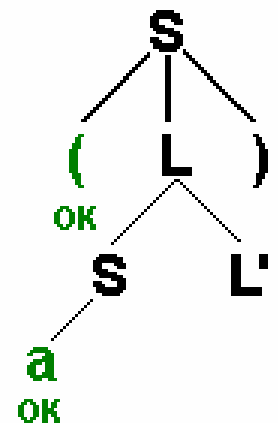


Pilha:

a
L'
)
\$



Árvore sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Consultar pos.[L',)] na tabela:

	a	,	(	)
S	$S \rightarrow a$		$S \rightarrow (L)$	
L	$L \rightarrow SL'$		$L \rightarrow SL'$	
L'		$L' \rightarrow ,SL'$		$L' \rightarrow \epsilon$

Pela regra acima, apagar L' da pilha:

Buffer de entrada:

(a) \$



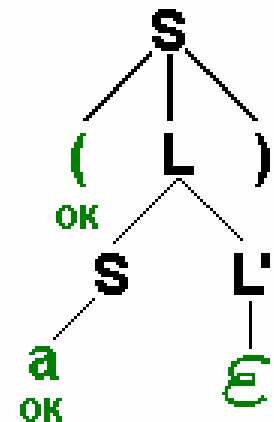
Pilha:

L'
)
\$

~~\$~~ ~~L~~ ~~,~~ ~~\$~~ ~~a~~ ~~L'~~

)
\$

Árvore sintática:



Exemplo de uso do analisador LL(1)

$$\begin{array}{c} S \rightarrow (L) \\ | \quad a \end{array}$$
$$L \rightarrow SL'$$
$$L' \xrightarrow{\epsilon} SL'$$

Teste de aceitação de cadeia:

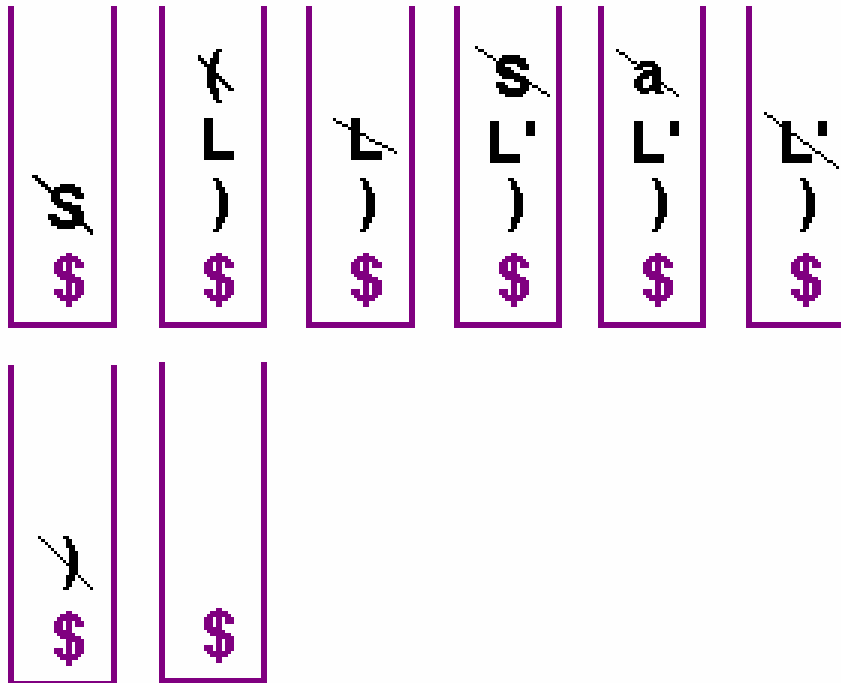
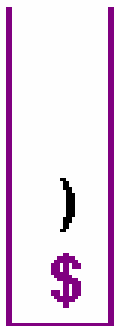
Casamento OK: topo de pilha e ponteiro de buffer de entrada com mesmo símbolo.

Eliminar) da pilha e andar com ponteiro do buffer :

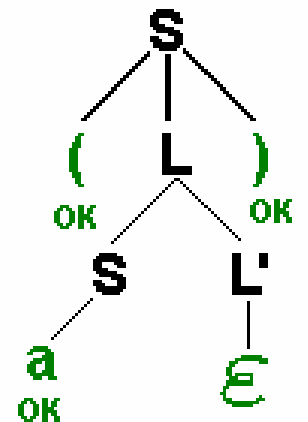
Buffer de entrada:



Pilha:



Árvore sintática:



Exemplo de uso do analisador LL(1)

$S \rightarrow (L)$

| a

$L \rightarrow SL'$

$L' \rightarrow ,SL'$

| ϵ

Teste de aceitação de cadeia:

Fim de análise: \$ no buffer e \$ na pilha.

Cadeia de entrada aceita OK.

Buffer de
entrada:

(	a	)	\$
---	---	---	----



Pilha:

\$

\$	L	)	\$	a	L'
\$	\$	\$	\$	\$	\$

)	\$
--------------	----

OK

Árvore
sintática:

