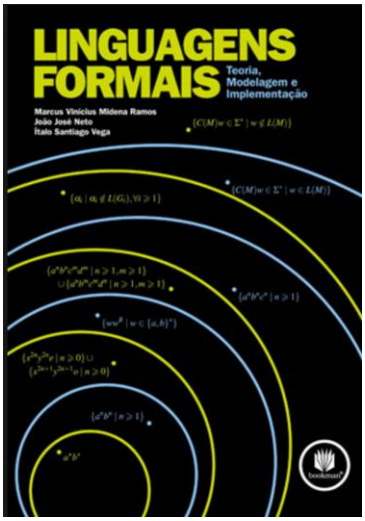
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
Linguagens Formais e Autômatos – Aula 10 – 1º SEMESTRE/2014
Prof. Luciano Silva

TEORIA: ANÁLISES DESCENDENTE E ASCENDENTE

Nossos **objetivos** nesta aula são:

- conhecer os processos de análise descendente a ascendente a partir de tabelas de análise prontas
- praticar com análises descendente e ascendente



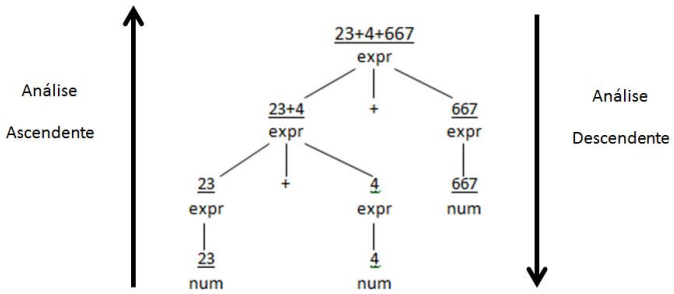
Para esta semana, usamos como referência as **Seções 4.1 (Gramáticas Livres de Contexto) até 4.4 (Ambigüidade)** do nosso livro da referência básica:

RAMOS, M.V.M., JOSÉ NETO, J., VEJA, I.S. **Linguagens Formais: Teoria, Modelagem e Implementação**. Porto Alegre: Bookman, 2009.

Não deixem de ler estas seções depois desta aula!

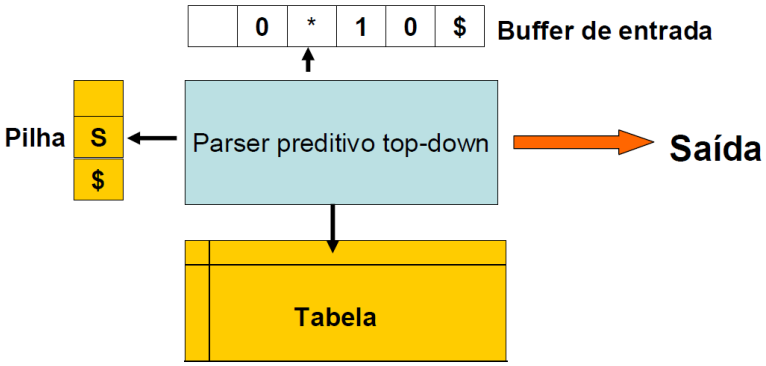
ANÁLISE DESCENDENTE

- Na nossa aula anterior, vimos processo de geração da árvore sintática é chamado de **análise sintática** e pode ser realizada de duas maneiras: de cima para baixo (**análise descendente ou *top-down***) ou de baixo para cima (**análise ascendente ou *bottom-up***) .



- Uma das maneiras de se realizar a análise descendente é via análise LL(k) (*Left-to-Right with Leftmost derivation*), que varre a entrada a ser processada da esquerda para a direita e realiza as derivações o mais à esquerda possível, utilizando k tokens para decidir qual regra aplicar. Uma categoria muito interessante dos analisadores LL são os analisadores LL(1), também conhecidos como *parsers* preditivos *top-down*, que utilizam apenas um token para decidir qual regra aplicar.

- O esquema básico de um analisador LL(1) é mostrado abaixo:



  - Para processar um buffer de entrada, o analisador utiliza uma tabela, chamada **Tabela de Análise LL(1)**, e uma pilha para guardar símbolos terminais e não-terminais. Este analisador é um tipo de **Autômato à Pilha**.
  - Vamos considerar, como exemplo, a seguinte gramática livre de contexto:

$$S \rightarrow cAa$$
$$A \rightarrow cB \mid B$$
$$B \rightarrow bcB \mid \varepsilon$$

  - A tabela de análise LL(1), denotada por M(N,T), é mostrada abaixo (veremos, em uma aula posterior, como construir tal tabela). Nas linhas, teremos sempre símbolos não-terminais e, nas colunas, teremos sempre símbolos terminais:

	a	b	c	$
S	ERRO	ERRO	$S \rightarrow cAa$	ERRO
A	$A \rightarrow B$	$A \rightarrow B$	$A \rightarrow cB$	ERRO
B	$B \rightarrow \varepsilon$	$B \rightarrow bcB$	ERRO	ERRO

- **Algoritmo de Análise LL(1):**

1. Inicializar a pilha com S$ (S, símbolo inicial da gramática, S no topo da pilha)
2. Repita os passos abaixo até reconhecer a entrada ou até encontrar um erro
3. Seja X o símbolo no topo da pilha e seja t o símbolo na entrada
4. Se X = $ e t = $, **entrada reconhecida**.
5. Se X = t e t≠$, desempilhar X e avançar um símbolo na entrada.
6. Se X for um símbolo não-terminal
7. Se M(X,t) contém ERRO, colocar estado do analisador em ERRO.
8. Senão, seja M(X,T)= X→ UVW. Desempilhar X e empilhar UVW, com U no topo da pilha.

- Exemplo de análise para a entrada **cbca$**:

Pilha	Entrada	Ação
S$	cbca$	S->cAa
cAa$	cbca$	casar c
Aa$	bca$	A->B
Ba$	bca$	B->bcB
bcBa$	bca$	casar b
cBa$	ca$	casar c
Ba$	a$	B-> ε
a$	a$	casar a
$	$	casar $, sucesso

EXERCÍCIO TUTORIADO

Utilizando a tabela de análise LL(1) anterior, simule o processo de análise para a entrada ccbca$:

Pilha	**Entrada**	**Ação**
S$	ccbca$	

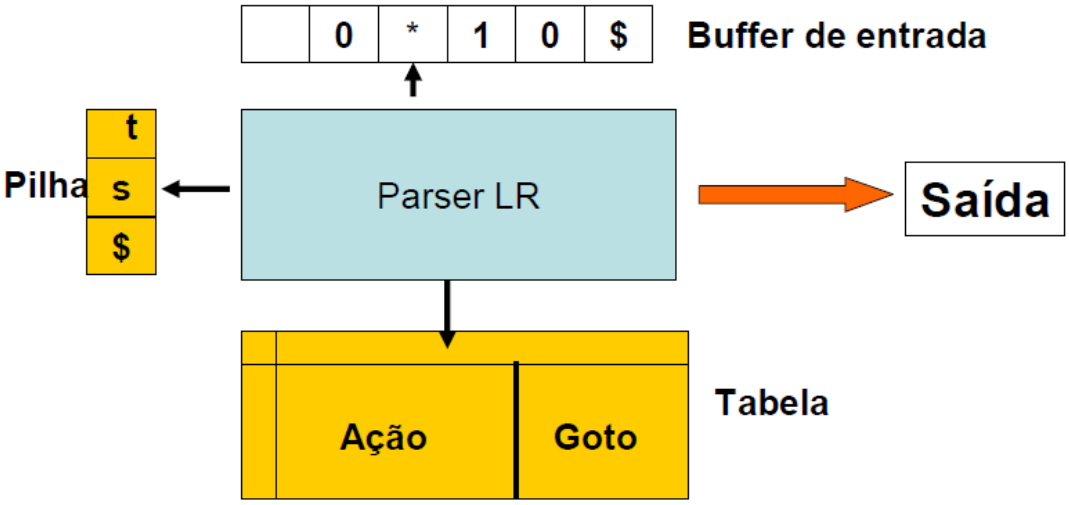
EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Utilizando a tabela de análise LL(1) anterior, simule o processo de análise para a entrada ca$:

Pilha	**Entrada**	**Ação**
S$	ca$	

- Uma das maneiras de se realizar a análise ascendente é via análise LR(k) (*Left-to-Right with Rightmost derivation*), que varre a entrada a ser processada da esquerda para a direita e realiza as derivações o mais à direita possível, utilizando k tokens para decidir qual regra aplicar.

- Existem diversas categorias de analisadores LR: LR(0), LR(1), SLR(1) e LALR(1). Porém, todos eles utilizam uma estrutura de análise comum, mostrada abaixo:



  - Para processar um buffer de entrada, o analisador utiliza uma tabela, chamada **Tabela de Análise LR**, formada de ações (**S**hift-empilha e **R**educe-reduz) e gotos (próximo estado que será empilhado) e uma pilha para guardar estados. Este analisador também é um tipo de **Autômato à Pilha**.

  - Para ilustrar como com funciona este esquema de análise, vamos considerar a seguinte gramática:

$$S \rightarrow T$$
$$T \rightarrow F \mid T * F$$
$$F \rightarrow id \mid (T)$$

- A tabela de análise LR(0) (ações e goto's) é mostrada abaixo. Veremos, em aulas posteriores, como construir esta tabela:

Productions	
1	T → F
2	T → T*F
3	F → id
4	F → (T)

	Ações					Goto		
	*	(	)	id	$	E	T	F
0		S5		S8			2	1
1	R1	R1	R1	R1	R1			
2	S3				Ok!			
3		S5		S8				4
4	R2	R2	R2	R2	R2			
5		S5		S8			6	1
6	S3		S7					
7	R4	R4	R4	R4	R4			
8	R3	R3	R3	R3	R3			

- Nesta tabela, a primeira coluna sempre indica os números dos estados. Nas **Ações**, sempre teremos **símbolos terminais** e, em **Goto**, símbolos **não-terminais**. **SN** indica que vamos empilhar o estado **N** e **RN** que vamos reduzir segundo a produção **N** (segundo a numeração de **Productions**). Espaços em branco em Ações denotam **ERRO**.

- **Algoritmo de Análise LR(0):**

1. Empilhar estado 0 na pilha.
2. Repetir os passos abaixo até que se encontre **OK!** ou **ERRO**
3. **Verificar qual ação tomar:**
4. Se Ações[Topo da Pilha,símbolo da entrada] não contém **ERRO**
5. Se encontramos OK!, acabamos de reconhecer a entrada.
6. Se ação é do tipo **SN**, empilhar estado N e avançar um símbolo na entrada
7. Se ação é do tipo **RN**, desempilhar todos os estados dos símbolos reduzidos segundo a regra N (Productions). Se Goto [Topo da Pilha, Símbolo Reduzido] está definido, empilhar estado Goto [Topo da Pilha, Símbolo Reduzido].
8. Senão, estamos em estado de **ERRO**.

- **Exemplo de análise LR(0) para (X)*Y$:**

Productions	
1	T → F
2	T → T*F
3	F → id
4	F → (T)

	Ações					Goto		
	*	(	)	id	$	E	T	F
0		S5		S8			2	1
1	R1	R1	R1	R1	R1			
2	S3				Ok!			
3		S5		S8				4
4	R2	R2	R2	R2	R2			
5		S5		S8			6	1
6	S3		S7					
7	R4	R4	R4	R4	R4			
8	R3	R3	R3	R3	R3			

Pilha	**Entrada**	**Ação/Goto**
0	(id)*id$	S5
0 5	id)*id$	S8
0 5 8	)*id$	R3 (F→id), desempilha 8, goto [5,F]=1
0 5 1	)*id$	R1 (T→F), desempilha 1, goto [5,T]=6
0 5 6	)*id$	S7
0 5 6 7	*id$	R4 (F→(T)), desempilha 7 6 5, goto [0,F]=1
0 1	*id$	R1 (T→F), desempilha 1, goto[0,T]=2
0 2	*id$	S3
0 2 3	id$	S8
0 2 3 8	$	R3 (F→id), desempilha 8, goto[3,F]=4
0 2 3 4	$	R2 (T→T*F), desempilha 4 3 2, goto[0,T]=2
0 2	$	**OK! ENTRADA ACEITA**

EXERCÍCIO TUTORIADO

Utilizando as tabelas **LR(0)** e **Productions** anteriores, mostre o processo de análise LR para a entrada SOMA*X$ (id*id$):

Pilha	Entrada	Ação/Goto
0	id*id$	

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Utilizando as tabelas **LR(0)** e **Productions** anteriores, mostre o processo de análise LR para a entrada (SOMA)$ ((id)$):

Pilha	Entrada	Ação/Goto
0	(id)$	

EXERCÍCIOS EXTRA-CLASSE

1. Considere a tabela LL(1) mostrada abaixo:

	a	b	c	$
S	ERRO	ERRO	**S → cAa**	ERRO
A	**A → B**	**A → B**	**A → cB**	ERRO
B	**B → ε**	**B →bcB**	ERRO	ERRO

Simule o processo de análise para a entrada caab$:

Pilha	**Entrada**	**Ação**
S$	caab$	

2. Utilizando as tabelas **LR(0)** e **Productions** abaixo, mostre o processo de análise LR para a entrada (X)(Y)$ ((id)*(id)$):

Productions	
1	T → F
2	T → T*F
3	F → id
4	F → (T)

	Ações					Goto		
	*	(	)	id	$	E	T	F
0		S5		S8			2	1
1	R1	R1	R1	R1	R1			
2	S3				Ok!			
3		S5		S8				4
4	R2	R2	R2	R2	R2			
5		S5		S8			6	1
6	S3		S7					
7	R4	R4	R4	R4	R4			
8	R3	R3	R3	R3	R3			

Pilha	Entrada	Ação/Goto
0	(id)*(id)$	