Universidade Cruzeiro do Sul

Linguagens Formais e Autômatos

Prof. Jose Claudio de Sousa
jcsousa@cruzeirodosul.edu.br

Gramática livre de contexto

- **Gramática livre de Contexto:** as gramáticas livres de contextos são importantes para definir linguagens de programação. Também são utilizadas em **editores de textos** e em aplicações de inteligência artificial. Nas linguagens de programação, por exemplo, as gramáticas livres de contexto são úteis no **balanceamento de parênteses** e nas **expressões aritméticas.**

- As gramáticas livres de contextos tem a principal forma

A -> α

- Caso ocorra uma derivação, a variável A que é o contexto, não depende de qualquer símbolo que a anteceda, ou a suceda, ou seja, é livre de contexto, levando o nome de livre de contexto.

Gramática livre de contexto

- **Definição formal**: Uma gramática livre de contexto pode ser representada por uma quadrupla G(V, T, P, S) onde:

–**V** é o conjunto finito dos símbolos **Não Terminais**.

–**T** é o conjunto finito dos **símbolos terminais** que correspondem ao **alfabeto da linguagem** definida pela gramática.

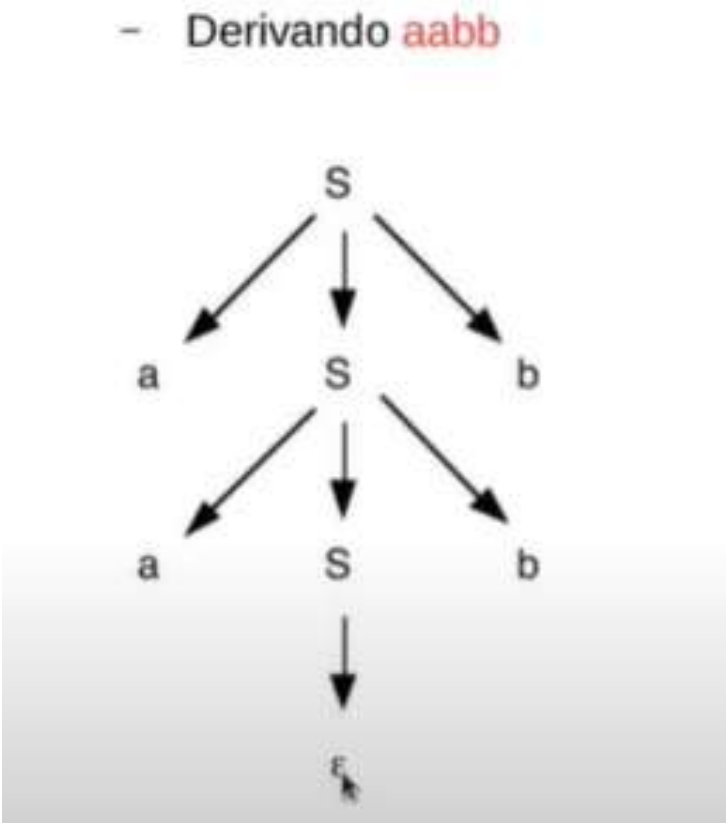
-**P** é o conjunto das **regras de produção** da gramática.

–**S** é **a raiz da gramática (variável inicial)**.

Árvore de derivação

- Algumas vezes pode ser útil **mostrar as derivações de uma GLC** através de **representações gráficas**. Essas representações são chamadas de **árvores de derivação** ou **árvores sintáticas** impõem **estruturas hierárquicas às sentenças das linguagens** geradas. **A árvore de derivação é a saída lógica da análise sintática**, constituindo uma representação intermediária utilizada na análise semântica. Em uma árvore de derivação;
    - a **raiz é o símbolo inicial** da gramática,
    - os **vértices interiores** (ou nós internos) são **variáveis (não terminais)** e os **vértices folhas** (ou nós folha) são símbolos terminais e não vazios.
- A árvore de Derivação proporciona uma melhor visualização da estrutura da sentença. Auxilia na demonstração formal de teoremas. Facilita a representação interna dos compiladores e interpretadores.

Árvore de derivação

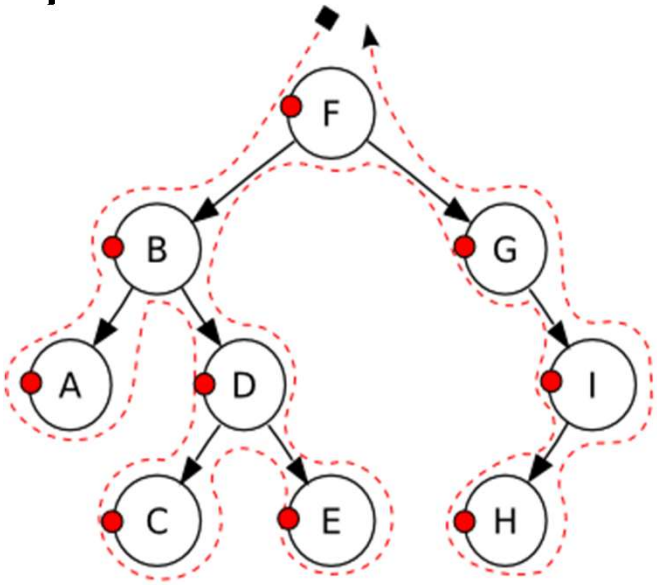


Percurso em árvore

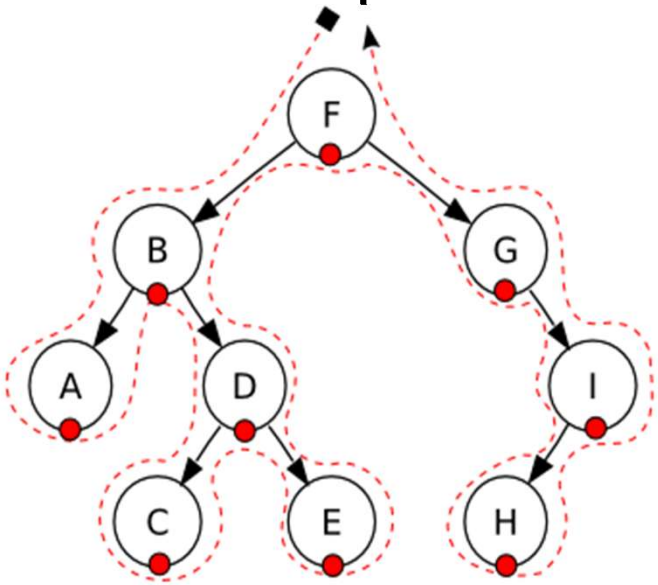
Pré-ordem: Visita primeiro a **raiz**, depois percorre recursivamente a **sub-árvore esquerda**, e por último a **sub-árvore direita**.

Em-ordem: visita primeiro a **sub-árvore esquerda**, depois a **raiz** e por último, a **sub-árvore à direita**.

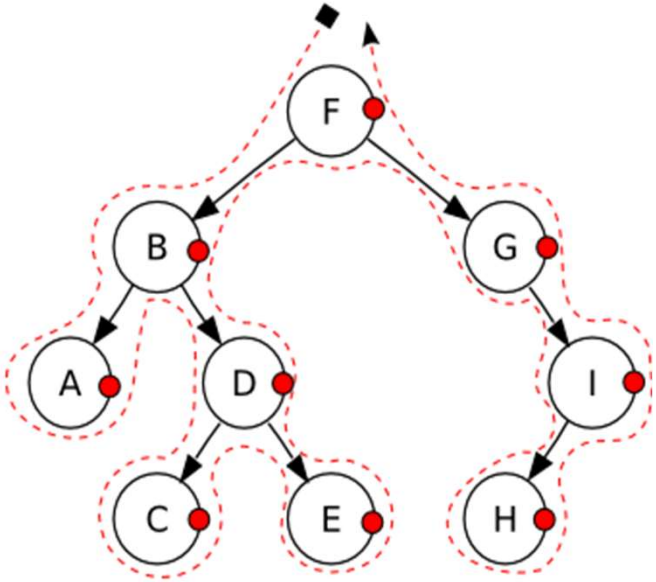
Pós-ordem: visita primeiro a, depois a **sub-árvore à direita** e por último a **raiz**. sub-árvore esquerda



Pré-ordem: F, B, A, D, C, E, G, I, H



Ordem simétrica: A, B, C, D, E, F, G, H, I



Pós-ordem: A, C, E, D, B, H, I, G, F

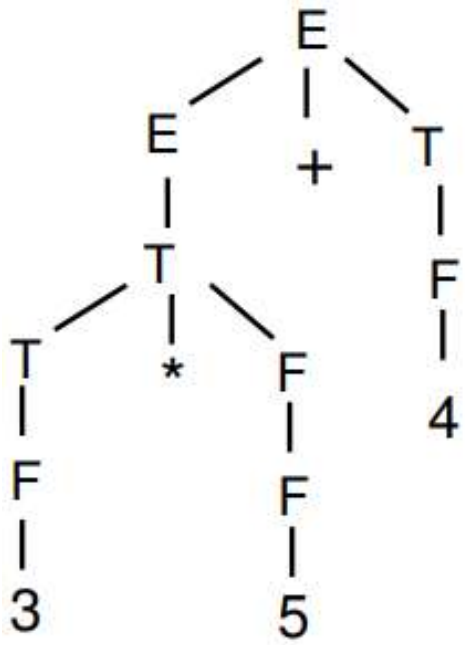
Árvore sintática abstrata

Árvore sintática abstrata

- Uma árvore sintática abstrata é uma árvore gramatical na qual:
  - Operadores e palavras chaves não aparecem como folhas, mas como nós interiores da árvore

Árvore sintática abstrata

- Uma árvore sintática abstrata é uma árvore gramatical na qual:
  - Operadores e palavras chaves não aparecem como folhas, mas como nós interiores da árvore



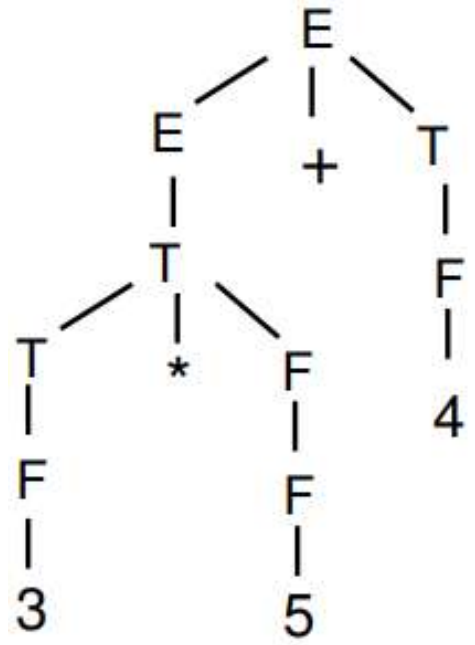
Árvore gramatical

Árvore sintática abstrata

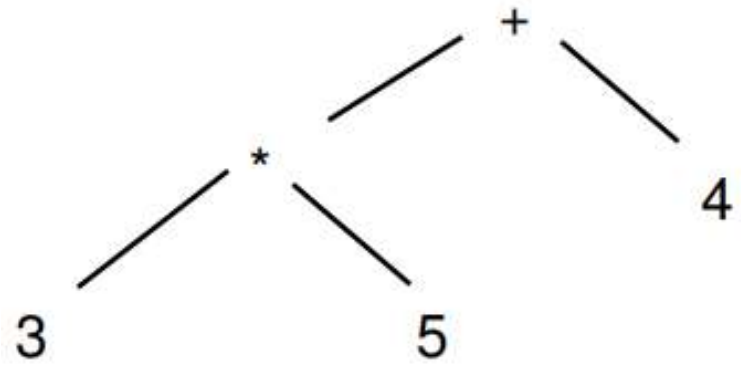
Uma árvore sintática abstrata é uma árvore gramatical na qual:

Operadores e palavras chaves não aparecem como folhas, mas como nós interiores da árvore

E -> E + T | T
T -> T * F | F
F -> F | 3 | 4 | 5



Árvore gramatical



Árvore sintática abstrata

Gramática ambígua

A partir de uma mesma palavra algumas vezes pode-se obter várias derivações de árvores devido à escolha do lado da derivação (esquerdo ou direito).

A derivação mais à esquerda de uma palavra;

- Sempre obtém o símbolo não terminal mais à esquerda.

A derivação mais à direita de uma palavra.

- Sempre obtém o símbolo não terminal mais à direita.

Caso seja possível numa mesma palavra obter duas ou mais árvores de derivação tem-se caracterizado uma gramática ambígua.

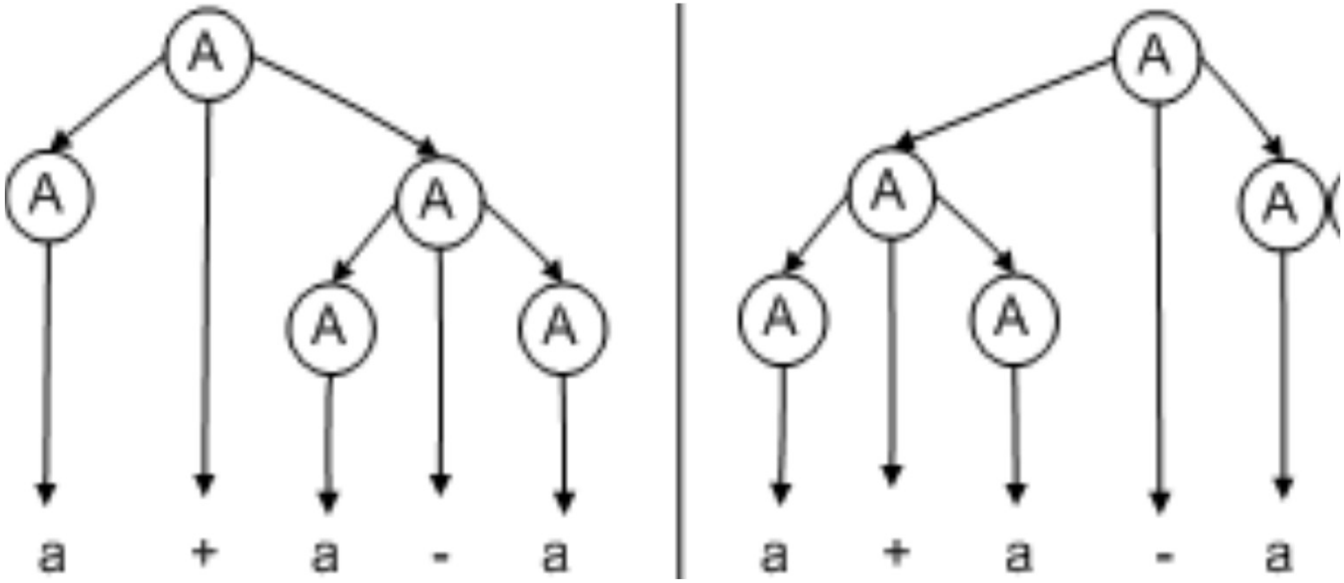
Universidade Cruzeiro do Sul

Gramática ambígua

Uma gramática é definida como ambígua se existem duas ou mais árvores de derivação para a mesma palavra.

Se quisermos derivar a + a - a, existem duas formas:

A → A + A | A – A | a



Notação BNF

Uma maneira usual de representar uma gramática livre de contexto é o uso da Forma **BNF**. Em uma BNF vale que:

As **variáveis não terminais** são palavras delimitadas pelos símbolos **<T>**.

As palavras **não delimitadas são terminais.**

A regra de produção **A → α** é substituído por **A ::= α**

O | é usado para separar os símbolos.

Ex:

F -> α
F -> E

<F> :: = α | (<E>)

Estratégias básicas para percorrer a árvore

Há duas estratégias básicas para percorrer a árvore:

- **Estratégia TOP-DOWN ou DESCENDENTE.**

```
<Frase> ::= <Sujeito> <Verbo> <Objeto>
<Sujeito> ::= "O gato"
<Verbo> ::= "comeu"
<Objeto> ::= "o peixe"
```

- **Estratégia BOTTOM-UP ou REDUTIVA.**

1. Começa com `"O gato comeu o peixe"`
2. Agrupa `"O gato"` → `<Sujeito>`, `"comeu"` → `<Verbo>`, `"o peixe"` → `<Objeto>`
3. Agrupa tudo → `<Frase>`

Estratégias

Os métodos de análise baseados na estratégia **top-down** (descendente) constroem a árvore de derivação a partir do símbolo inicial da gramática (raiz da árvore), fazendo a árvore crescer até atingir suas folhas.

A estratégia **bottom-up** (redutiva) realiza a análise no sentido inverso, isto é, a partir das palavras do texto fonte (folhas da árvore de derivação) constrói a árvore até o símbolo inicial da gramática.

Na estratégia **top-down**, em cada passo, um lado esquerdo de produção é substituído por um lado direito (expansão); na estratégia **bottom-up**, em cada passo, um lado direito de produção é substituído por um símbolo não-terminal (redução).

Análise sintática descendente (TOP-DOWN).

- Um algoritmo para análise sintática **descendente** analisa a cadeia de marcas de entrada pelo acompanhamento dos passos de uma derivação à esquerda.
- O nome "descendente" vem da forma como a árvore de análise sintática é percorrida – em pre-ordem, portanto, da raiz para as folhas.
- Há duas formas de analisadores sintáticos descendentes:
  -analisadores com retrocesso e
  -analisadores preditivos.
- Um analisador **sintático preditivo** tenta prever a construção seguinte na cadeia de entrada com base em uma ou mais marcados de verificação à frente.
- Já um analisador sintático com **retrocesso** testa diferentes possibilidades de análise sintática da entrada, retrocedendo se alguma possibilidade falhar.

Análise sintática descendente (TOP-DOWN).

- Os dois tipos de algoritmos mais comuns para análise sintática descendente são;
  - **descendentes recursivos** e o
  - algoritmo **LL(1)**.
- A análise sintática descendente recursiva é bastante versátil e é o método mais adequado para um analisador sintático escrito manualmente.
- O método de **análise sintática LL(1)** tem esse nome pelo seguinte: o primeiro "L" se refere ao fato de o processamento ocorrer da esquerda para a direita(esquerda em Inglês *left*).
- O segundo "L" se refere ao fato de o analisador acompanhar uma derivação à esquerda para a cadeia de entrada.
- O número 1 entre parênteses significa que ela usa apenas um símbolo da entrada para prever a direção da análise.
- Podemos ter também, a análise sintática LL(K), que realiza a verificação à frente de K símbolos, mas apenas um símbolo para verificação à frente é o caso mais comum.

Análise recursiva com retrocesso.

- Ex: considere a sentença [a] derivada a partir da gramática abaixo:

S -> a | [L]
L-> S; L | S

- A análise descendente dessa sentença começa com o símbolo inicial S na raiz da árvore de derivação.

- A primeira produção aplicada dever ser S -> [L]

Análise recursiva com retrocesso.

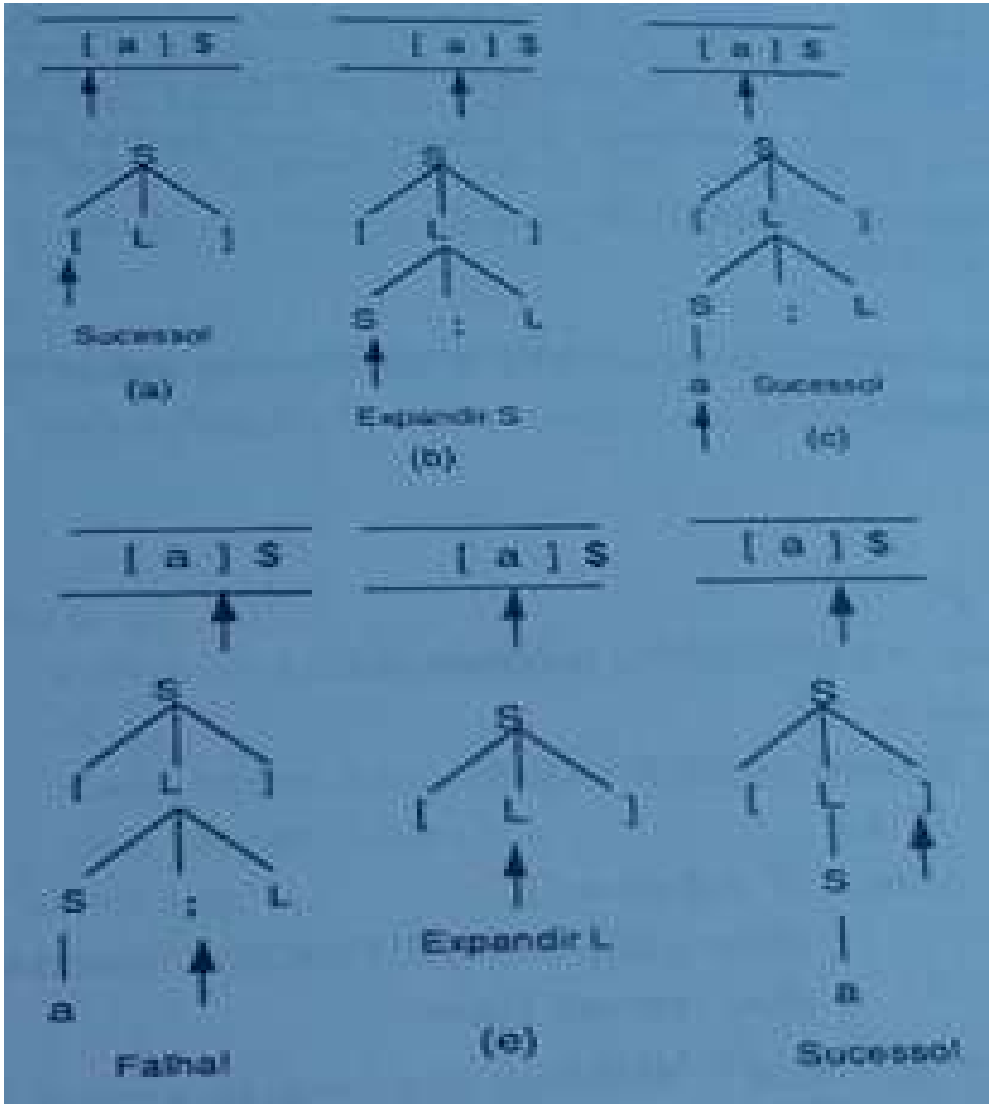
- Esta análise faz a expansão da árvore de derivação a partir da raiz, expandindo sempre o não-terminal mais à esquerda.

- Quando existe mais de uma regra de produção para o não-terminal a ser expandindo, a opção escolhida é a função da próxima palavra.

- Se a palavra de entrada não define a produção a ser usada, então todas as alternativas vão ser testadas até que se obtenha sucesso (ou até que a análise falhe irremediavelmente).

Universidade Cruzeiro do Sul

Análise recursiva com retrocesso.



Análise recursiva com retrocesso.

- O reconhecimento de [é bem sucedido, e a análise prossegue com a derivação de L, que pode ser efetuada usando um dos lados direitos alternativos: S ; L ou S.
- No exemplo, é escolhida a primeira alternativa S ; L e S é expandido novamente, obtendo-se sucesso.
- Porém a comparação seguinte (] com ;) falha e o analisador deve, então, retroceder na fita de entrada para o ponto em que esta estava por ocasião da opção pela primeira alternativa de L.
- É aplicada, então, a segunda alternativa, L -> S.
- A derivação final é obtida aplicando-se a produção S -> a.

Algoritmo LL(1)

- A análise sintática LL(1) utiliza uma pilha explícita, em vez de recursividade.
- A representação dessa pilha de forma padrão é útil para facilitar e agilizar a visualização das ações do analisador sintático LL(1). Este algoritmo utiliza a tabela abaixo.

Pilha	Entrada	Ação
$ S	()	

Algoritmo LL(1)

•Neste exemplo, o símbolo de início é **S** e cadeia de entrada é **( )**

•Um analisador sintático descendente substitui um não-terminal no topo da pilha por uma de suas regras de produções.

• Nesta substituição, caso haja o casamento do símbolo terminal com o símbolo de entrada ambos são descartados, tanto na pilha como na entrada.

• Dessa forma, essas duas opções são possíveis;

a) Substituir um não-terminal A no topo da pilha por uma cadeia α com base na escolha gramatical A -> α.

b) Casar um terminal que esteja no topo da pilha com o símbolo seguinte da entrada.

Algoritmo LL(1)

- Com um não-terminal *A* no topo da pilha, o uso do método de análise sintática LL(1) exige uma escolha, baseada no símbolo de entrada (verificação a frente) da regra gramatical para que *A* seja utilizado na substituição de *A* na pilha.

- Entretanto, não é necessário nenhuma escolha quando há um terminal no topo da pilha, pois ela é igual ou diferente do símbolo de entrada e neste caso, há um casamento, ou há um erro.

Algoritmo LL(1)

- Nesta tabela, a primeira coluna é a pilha, a segunda coluna é a entrada que será processada, ou seja, a palavra que será processada e a terceira coluna, é a ação pode ser realizada.

Pilha	Entrada	Ação
$ S	()	

Algoritmo LL(1)

- Este algoritmo é um algoritmo Descendente. A ideia é sair da raiz e chegar nas folhas. Com isso, a raiz já começa na pilha.
- O $ delimita o final da pilha.

Pilha	Entrada	Ação
$ S	()	

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε
$ S)	)	

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε
$ S)	)	Casamento

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε
$ S)	)	Casamento
$ S		

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε
$ S)	)	Casamento
$ S		S -> ε

Algoritmo LL(1)

- Regras
  - S -> (S) S
  - S -> ε

Pilha	Entrada	Ação
$ S	()	S -> (S) S
$ S) S (	()	Casamento
$ S) S	)	S -> ε
$ S)	)	Casamento
$ S		S -> ε
$		aceita

Perguntas?

Fim

:wq

Universidade Cruzeiro do Sul

Linguagens Formais e Autômatos

Prof. Jose Claudio de Sousa
jcsousa@cruzeirodosul.edu.br