Árvores de Análise Sintática

Definição

Derivações

Ambiguidade em Gramáticas

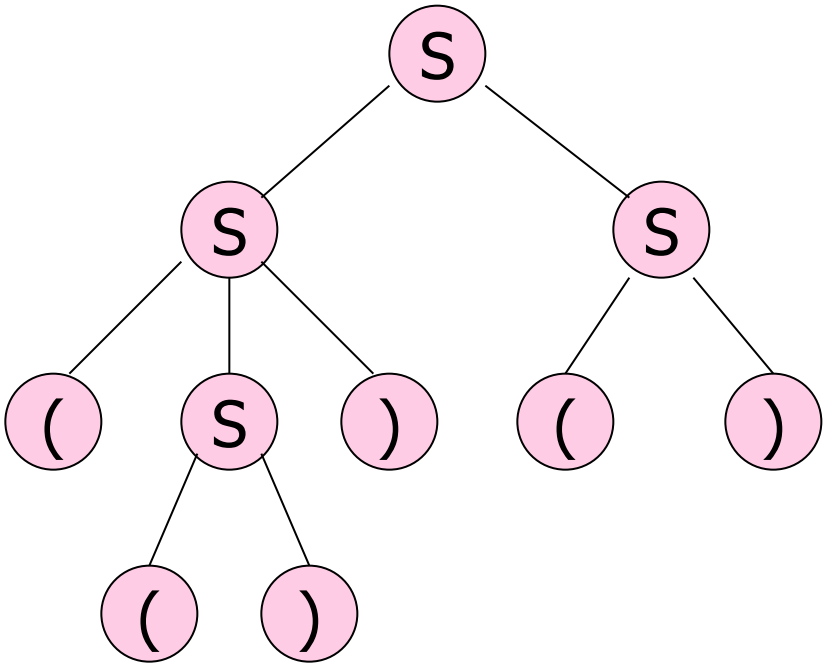
Tradução dos slides do Prof. Jeffrey D. Ullman (Stanford University)

Árvores de Análise Sintática

- *Árvores de Análise Sintática (Parse trees)* são árvores rotuladas por símbolos de uma CFG.
- Folhas: rotuladas por um terminal ou ϵ.
- Nó Interior: rotulado por uma variável.
  - Filhos são rotulados pelo lado direito de uma produção de seu pai.
- Raiz: precisa ser rotulada pelo símbolo inicial.

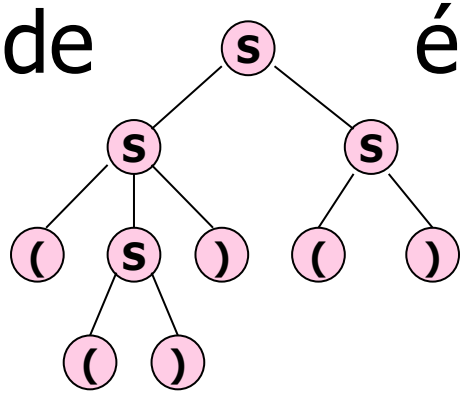
Exemplo: Árvore de Análise Sintática

S -> SS | (S) | ()



Resultado de uma Árvore de Análise Sintática

- A concatenação dos rotulos das folhas da esquerda para a direita
  - Istó é, percorrer a árvore em pré-ordem.

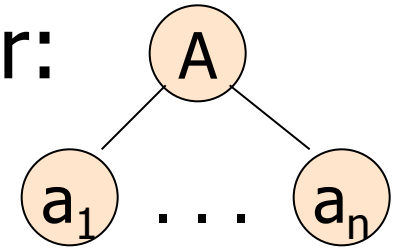
  é chamado o *resultado* de uma árvore de análise sintática.
- Exemplo: o resultado de  é (())()

Árvore de Análise Sintática e Derivações mais a esquerda/direita

- Para toda Árvore de Análise Sintática, existe uma única derivação mais a esquerda e uma única mais a direita.
- Prova:
  1. Se existe uma Árvore de Análise Sintática com raiz A e resultado w, então $A =>^*_{lm} w$.
  2. Se $A =>^*_{lm} w$, então existe uma Árvore de Análise Sintática com raiz A e resultado w.

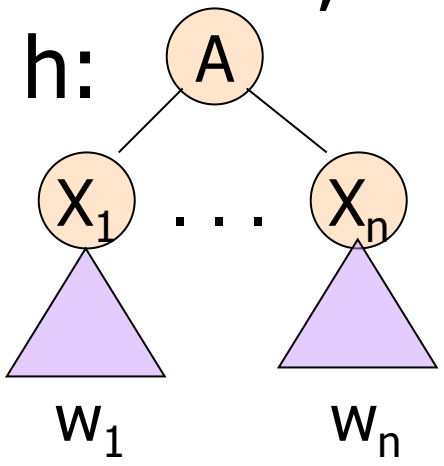
Prova – Parte 1

- Indução sobre a *altura* da árvore (comprimento do maior caminho a partir da raiz).
- Base: Altura 1. Árvore deve ser:
- A -> $a_1 \ldots a_n$ deve ser uma produção.
- Assim, A $=>_{lm}$ $a_1 \ldots a_n$.



Parte 1 – Indução

- Suponha (1) para árvores de altura < h, e permita esta árvore ter altura h:
- Por HI, $X_i =>^*_{lm} w_i$.
  - Note: se X_i é um terminal, então $X_i = w_i$.



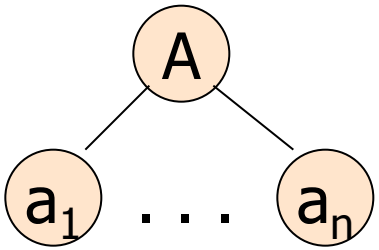
- Assim, $A =>_{lm} X_1 \ldots X_n =>^*_{lm} w_1 X_2 \ldots X_n =>^*_{lm} w_1 w_2 X_3 \ldots X_n =>^*_{lm} \ldots =>^*_{lm} w_1 \ldots w_n$

Prova: Parte 2

- Dada uma derivação mais à esquerda de uma string de terminal, precisamos provar a existência de uma árvore de análise sintática.
- A prova é uma indução sobre o comprimento da derivação.

Parte 2 – Base

- Se $A =>^*_{lm} a_1 \ldots a_n$ por uma derivação de uma etapa, então existe uma árvore de análise sintática

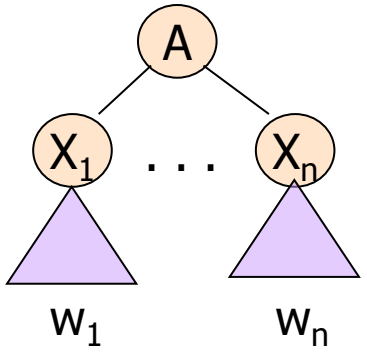


Parte 2 – Indução

- Suponha (2) para derivações de menos de k etapas (k > 1), e deixe $A =>^*_{lm} w$ ser uma derivação de k-etapas.
- Primeira etapa é $A =>_{lm} X_1 \ldots X_n$.
- Ponto chave: w pode ser dividido de modo a primeira porção ser derivada de X1, a próxima de X2, e assim por diante.
  - Se X_i é um terminal, então $w_i = X_i$.

Indução – (2)

- Isto é, $X_i =>^*_{lm} w_i$ para todo i tal que Xi é uma variável.
  - E a derivação leva menos que k-etapas.
- Pela HI, se X_i é uma variável, então existe uma árvore de análise sintática com raiz X_i e resultado w_i.
- Assim, existe um árvore de análise sintática.



Árvore de Análise Sintática e Derivações mais a direita

- As ideias são, essencialmente, a imagem da prova para as derivações mais à esquerda.

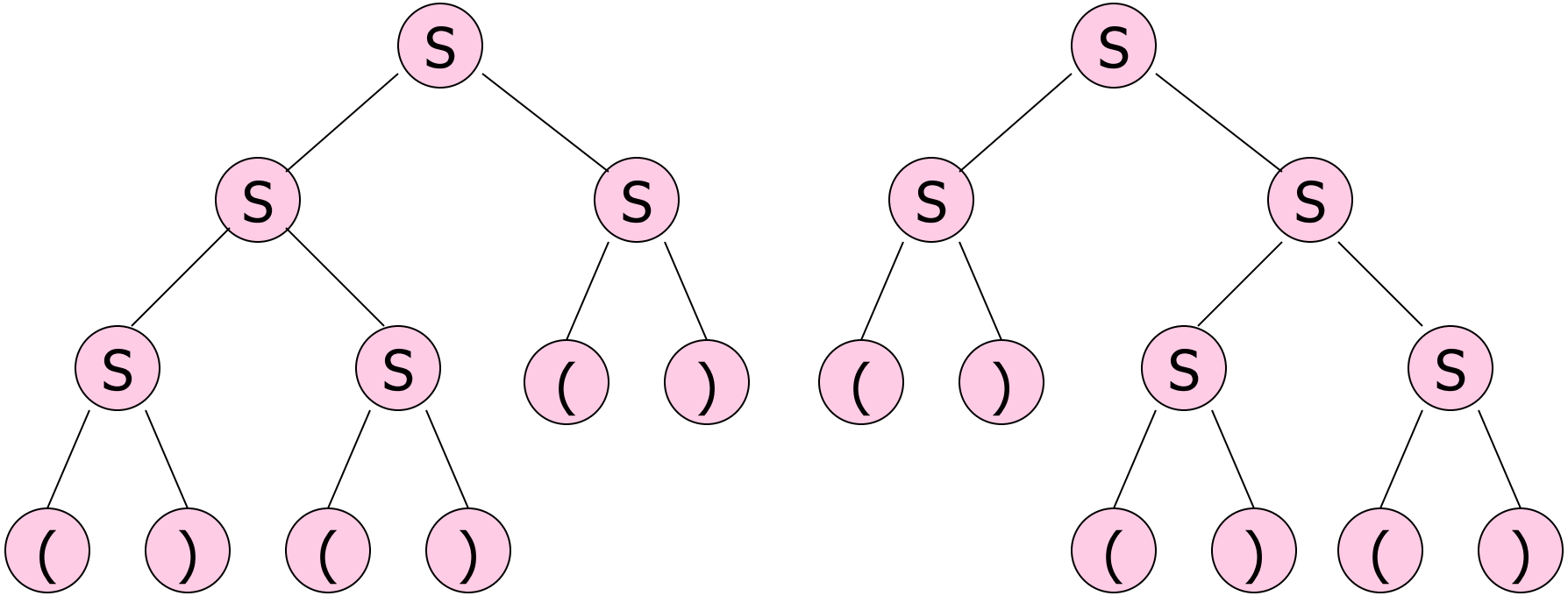
Árvore de Análise Sintática e Qualquer Derivações

- A prova que pode-se obter uma árvore de análise sintática a partir da derivação mais à esquerda realmente não depende do "mais a esquerda".
- A primeira etapa ainda tem que ser A => $X_1 \ldots X_n$.
- E w ainda pode ser dividido de modo a primeira porção ser derivada de X_1, a próxima é derivada de X_2, e assim por diante.

Gramáticas Ambíguas

- Uma CFG é *ambígua* se existe um string na linguagem que é o resultado de duas ou mais árvores de análise sintática.
- Exemplo: S -> SS | (S) | ()

Exemplo – Continuação



Ambiguidade, Derivações mais à esquerda/direita

- Se há duas árvores de análise sintática diferentes, elas devem produzir duas derivações mais à esquerda diferentes.
- Por outro lado, duas derivações mais à esquerda diferentes produzem árvores de análise sintática diferentes.
- Da mesma forma para as derivações mais à direita.

Ambiguidade, etc. – (2)

- Assim, as definições equivalentes de "gramática ambígua" são:
  1. Existe um string na linguagem que tem duas derivações mais à esqueda diferentes.
  2. Existe um string na linguagem que tem duas derivações mais à direita diferentes.

Ambiguidade é uma Propriedade de Gramáticas, não Linguagens

- Para a linguagem de parênteses balanceados, existe uma uma outra CFG que é não ambígua.

B -> (RB | ϵ

R ->) | (RR

B, o símbolo inicial, deriva strings balanceados .

R gera strings que tem um parênteses a mais à direita que à esqueda.

Exemplo: Gramática não-ambígua

B -> (RB | ϵ R ->) | (RR

- Construir uma única derivação mais à esquerda para um dado string de parênteses balanceado verificando o string da esquerda para a direita.
  - Se precisamos expandir B, então use B -> (RB se o próximo símbolo é "(" e ϵ se está no final.
  - Se precisamos expandir R, use R ->) se o próximo símbolo é ")" e (RR se este é "(".

O Processo de análise

Entrada restante:

(())()

Próximo símbolo

Etapas da derivação mais à equerda:

B

B -> (RB | ϵ R ->) | (RR

O Processo de análise

Entrada restante:

())()

Próximo símbolo

Etapas da derivação mais à equerda:

B

(RB

B -> (RB | ϵ R ->) | (RR

O Processo de análise

Entrada restante:

))()

Próximo símbolo

Etapas da derivação mais à equerda:

B

(RB

((RRB

B -> (RB | ϵ R ->) | (RR

O Processo de análise

Entrada restante:

)()

Próximo símbolo

Etapas da derivação mais à equerda:

B

(RB

((RRB

(()RB

B -> (RB | ϵ

R ->) | (RR

O Processo de análise

Entrada restante:

()

Próximo símbolo

Etapas da derivação mais à equerda:

B

(RB

((RRB

((()RB

((())B

B -> (RB | ϵ

R ->) | (RR

O Processo de análise

Entrada restante:

)

Próximo símbolo

Etapas da derivação mais à equerda:

B ((())(RB

(RB

((RRB

((()RB

((())B

B -> (RB | ϵ R ->) | (RR

O Processo de análise

Entrada restante:

Próximo símbolo

Etapas da derivação mais à equerda:

B	(())(RB
(RB	(())()B
((RRB	
(()RB	
(())B	

B -> (RB | ϵ

R ->) | (RR

O Processo de análise

Entrada restante:	Etapas da derivação mais à equerda:
B	(())(RB
(RB	(())()B
((RRB	(())()
(()RB	
(())B	
B -> (RB | ϵ	R ->) | (RR

Gramáticas LL(1)

- Uma gramática como B -> (RB | ϵ
  R ->) | (RR, onde pode-se sempre descobrir a produção para usar em uma derivação mais à esquerda verificando a string dada da esquerda para a direita e olhando apenas para o símbolo seguinte é chamada LL(1).
  - "Leftmost derivation, left-to-right scan, one symbol of lookahead."

Gramáticas LL(1)– (2)

- Muitas linguagens de programação tem gramáticas LL(1).
- Gramáticas LL(1) nunca são ambíguas.

Ambiguidade Inerente

- Seria bom se para cada gramática ambígua, houvesse uma forma de "consertar" a ambiguidade, como fizemos com a gramática de parênteses balanceados.
- Infelizmente, algumas CFL's são *inerentemente ambiguas*
  - Todas as gramáticas para a linguagem são ambíguas.

Exemplo: Ambiguidade Inerente

- A linguagem $\{0^i1^j2^k \mid i = j \text{ ou } j = k\}$ é inerentemente ambígua.
- Intuitivamente, pelo ao menos alguns dos strings da forma $0^n1^n2^n$ precisam ser gerados por duas diferentes árvores de análise sintática, uma baseada na verificação de 0's e 1's, e outra baseada na verificação de 1's e 2's.

Uma possível Gramática Ambígua

S -> AB | CD

A -> 0A1 | 01 — A gera igual 0's e 1's

B -> 2B | 2 — B gera algum número de 2's

C -> 0C | 0 — C gera algum número de 0's

D -> 1D2 | 12 — D gera igual 1's e 2's

Há duas derivações de todo string com igual números de 0's, 1's e 2's. Ex.:
S => AB => 01B =>012
S => CD => 0D => 012