



Universidade Federal do ABC

CMCC

Centro de Matemática, Computação e Cognição



Teoria da Computação

Gramáticas e Linguagens Livres de contexto

Mirtha Lina Fernández Venero
mirtha.lina@ufabc.edu.br

outubro 2017



Sumário

Gramáticas e Linguagens Livres de contexto

Árvore de Derivação

Gramáticas ambíguas

Análise Sintática (Parsing)

Bibliografia



Re-lembrando Gramáticas

Gramáticas: (N, Σ, S, P)

1. N (ou V) conjunto *finito* de símbolos **não-terminais***
2. Σ (ou T) conjunto *finito* de símbolos **terminais**
3. $S \in N$ **símbolo de começo**
4. $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ conjunto de **produções** ou **regras sintáticas** escritas $\alpha \rightarrow \beta$ ($\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$)

Exemplo:

$$G_1 = (\{A, S\}, \{0, 1, \dots, 9\}, S, \{S \rightarrow AS | A, A \rightarrow 0 | 1 | \dots | 9\})$$

$$G_2 = (\{A, S\}, \{0, 1\}, S, \{S \rightarrow 0A1, 0A \rightarrow 00A1, A \rightarrow \varepsilon\})$$

$$G_3 = S \rightarrow aSBc, S \rightarrow abc, cB \rightarrow Bc, bB \rightarrow bb$$



Re-lembrando Gramáticas

Gramáticas Livres de Contexto: (N, Σ, S, P)

1. N (ou V) conjunto *finito* de símbolos **não-terminais***
2. Σ (ou T) conjunto *finito* de símbolos **terminais**
3. $S \in N$ **símbolo de começo**
4. $P \subseteq N \times (N \cup \Sigma)^*$ conjunto de **produções ou regras sintáticas** escritas $A \rightarrow \beta$ ($A \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$)

Exemplo:

$$G_1 = (\{A, S\}, \{0, 1, \dots, 9\}, S, \{S \rightarrow AS | A, A \rightarrow 0 | 1 | \dots | 9\})$$

$$G_2 = (\{A, S\}, \{0, 1\}, S, \{S \rightarrow 0A1, 0A \rightarrow 00A1, A \rightarrow \varepsilon\})$$

$$G_3 = S \rightarrow aSBc, S \rightarrow abc, cB \rightarrow Bc, bB \rightarrow bb$$



Gramática Livre de Contexto para Expressões

Exemplo: Pertence a cadeia **id + id * id** à seguinte gramática?

$$S \rightarrow E$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Que derivação escolher para reconstruir a estrutura sintática do programa fonte? $\gamma \mathbf{A} \eta \rightarrow_G \gamma \beta \eta$ sse $\mathbf{A} \rightarrow \beta \in P$



Gramática Livre de Contexto para Expressões

Exemplo: Pertence a cadeia **id + id * id** à seguinte gramática?

$$S \rightarrow E$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Que derivação escolher para reconstruir a estrutura sintática do programa fonte? $\gamma \mathbf{A} \eta \rightarrow_G \gamma \beta \eta$ sse $\mathbf{A} \rightarrow \beta \in P$

- ▶ **Mais à esquerda:** $\mathbf{w} \mathbf{A} \eta \rightarrow_G \mathbf{w} \beta \eta$, $\mathbf{w} \in \Sigma^*$
- ▶ **Mais à direita:** $\gamma \mathbf{A} \mathbf{w} \rightarrow_G \gamma \beta \mathbf{w}$, $\mathbf{w} \in \Sigma^*$
- ▶ Arbitrária



Gramática Livre de Contexto para Expressões

Exemplo: Pertence a cadeia **id + id * id** à seguinte gramática?

$$S \rightarrow E$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Que derivação escolher para reconstruir a estrutura sintática do programa fonte? $\gamma \mathbf{A} \eta \rightarrow_G \gamma \beta \eta$ sse $\mathbf{A} \rightarrow \beta \in P$

Mais à esquerda

$$\begin{aligned} S &\rightarrow E \rightarrow \underline{E} + E \\ &\rightarrow \text{id} + \underline{E} \\ &\rightarrow \text{id} + \underline{E} * E \\ &\rightarrow \text{id} + \text{id} * \underline{E} \\ &\rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$

Mais à direita

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \\ &\rightarrow E + E * \underline{E} \\ &\rightarrow E + \underline{E} * \text{id} \\ &\rightarrow \underline{E} + \text{id} * \text{id} \\ &\rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$

Arbitrária

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \\ &\rightarrow \underline{E} + E * E \\ &\rightarrow \text{id} + \underline{E} * E \\ &\rightarrow \text{id} + \text{id} * \underline{E} \\ &\rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$



Gramática Livre de Contexto para Expressões

Exemplo: Pertence a cadeia **id + id * id** à seguinte gramática?

$$S \rightarrow E$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Que derivação escolher para reconstruir a estrutura sintática do programa fonte? $\gamma \mathbf{A} \eta \rightarrow_G \gamma \beta \eta$ sse $\mathbf{A} \rightarrow \beta \in P$

- ▶ **Mais à esquerda:** $\mathbf{w} \mathbf{A} \eta \rightarrow_G \mathbf{w} \beta \eta$, $\mathbf{w} \in \Sigma^*$
- ▶ **Mais à direita:** $\gamma \mathbf{A} \mathbf{w} \rightarrow_G \gamma \beta \mathbf{w}$, $\mathbf{w} \in \Sigma^*$
- ▶ Arbitrária

Diferentes derivações podem usar as mesmas produções, porém aplicá-las numa ordem diferente. Por que não usar uma árvore?



Sumário

Gramáticas e Linguagens Livres de contexto

Árvore de Derivação

Gramáticas ambíguas

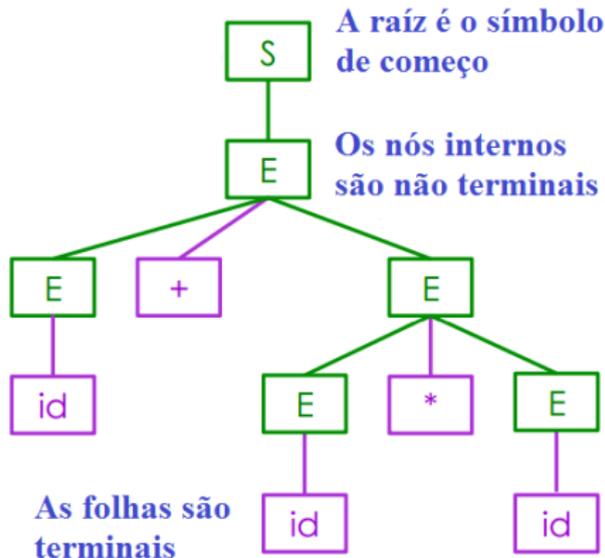
Análise Sintática (Parsing)

Bibliografia

Derivações Canônicas e Árvore de Derivação

$$\begin{aligned} S &\rightarrow E \rightarrow \underline{E} + E \rightarrow id + \underline{E} \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow E + E * \underline{E} \\ &\rightarrow E + \underline{E} * id \rightarrow \underline{E} + id * id \\ &\rightarrow id + id * id \end{aligned}$$

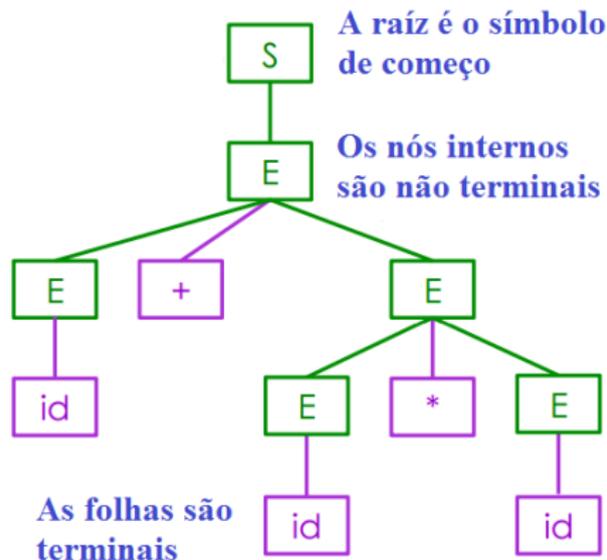
$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow \underline{E} + E * E \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$


Derivações Canônicas e Árvore de Derivação

$$\begin{aligned} S &\rightarrow E \rightarrow \underline{E} + E \rightarrow id + \underline{E} \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow E + E * \underline{E} \\ &\rightarrow E + \underline{E} * id \rightarrow \underline{E} + id * id \\ &\rightarrow id + id * id \end{aligned}$$

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow \underline{E} + E * E \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$



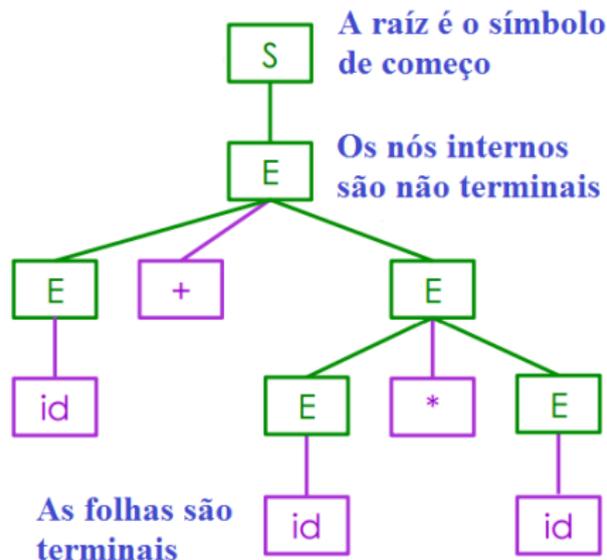
A produção aplicada gera os filhos cada nó interno da árvore

Derivações Canônicas e Árvore de Derivação

$$\begin{aligned} S &\rightarrow E \rightarrow \underline{E} + E \rightarrow id + \underline{E} \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow E + E * \underline{E} \\ &\rightarrow E + \underline{E} * id \rightarrow \underline{E} + id * id \\ &\rightarrow id + id * id \end{aligned}$$

$$\begin{aligned} S &\rightarrow E \rightarrow E + \underline{E} \rightarrow \underline{E} + E * E \\ &\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E} \\ &\rightarrow id + id * id \end{aligned}$$



A palavra gerada é a concatenação das folhas de esquerda a direita



Derivações Canônicas e Árvore de Derivação

- ▶ Uma árvore de derivação representa um conjunto de derivações que todas geram a mesma cadeia
- ▶ Uma árvore de derivação representa uma única derivação mais à esquerda e uma única derivação mais à direita
- ▶ Uma gramática pode ter **duas árvores diferentes** (derivações mais à esquerda-direita) para a mesma cadeia?

Exemplo: Pertence a cadeia **id + id * id** à gramática?

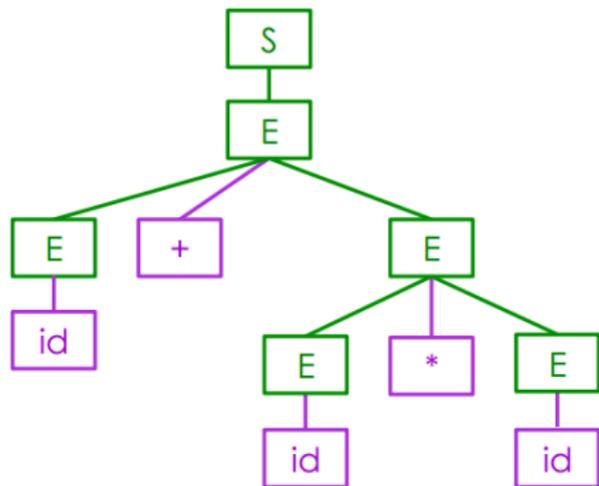
$$S \rightarrow E$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

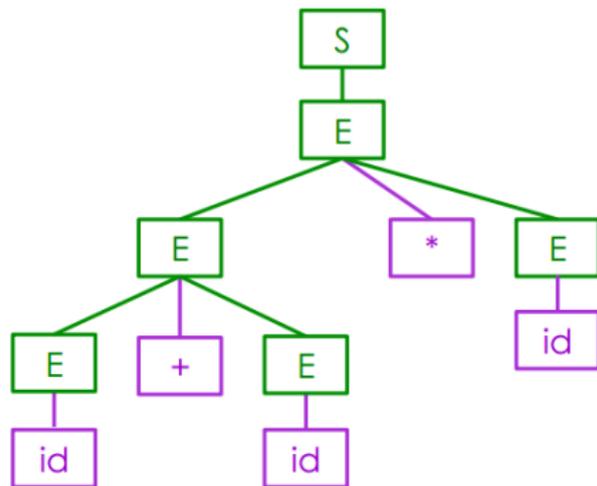


Derivações Canônicas, Árvore de Derivação, Ambiguidade

$S \rightarrow E \rightarrow \underline{E} + E \rightarrow id + \underline{E}$
 $\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E}$
 $\rightarrow id + id * id$



$S \rightarrow E \rightarrow \underline{E} * E \rightarrow \underline{E} + E * E$
 $\rightarrow id + \underline{E} * E \rightarrow id + id * \underline{E}$
 $\rightarrow id + id * id$





Sumário

Gramáticas e Linguagens Livres de contexto

Árvore de Derivação

Gramáticas ambíguas

Análise Sintática (Parsing)

Bibliografia

Ambiguidade nas linguagens humanas

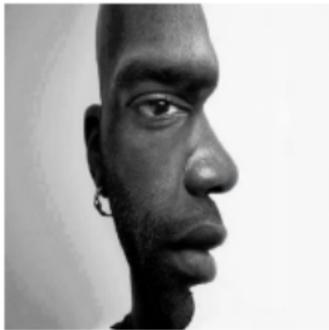
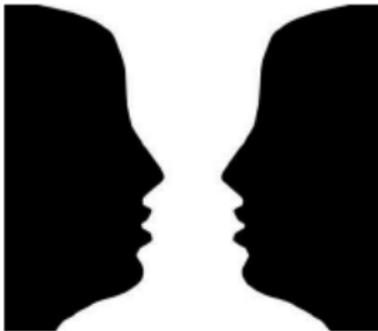
Duas interpretações diferentes para a mesma sentença/imagem

I saw the man on the hill with a telescope.

John saw my dog driving to work this morning.

Eu li a notícia sobre a greve na faculdade.

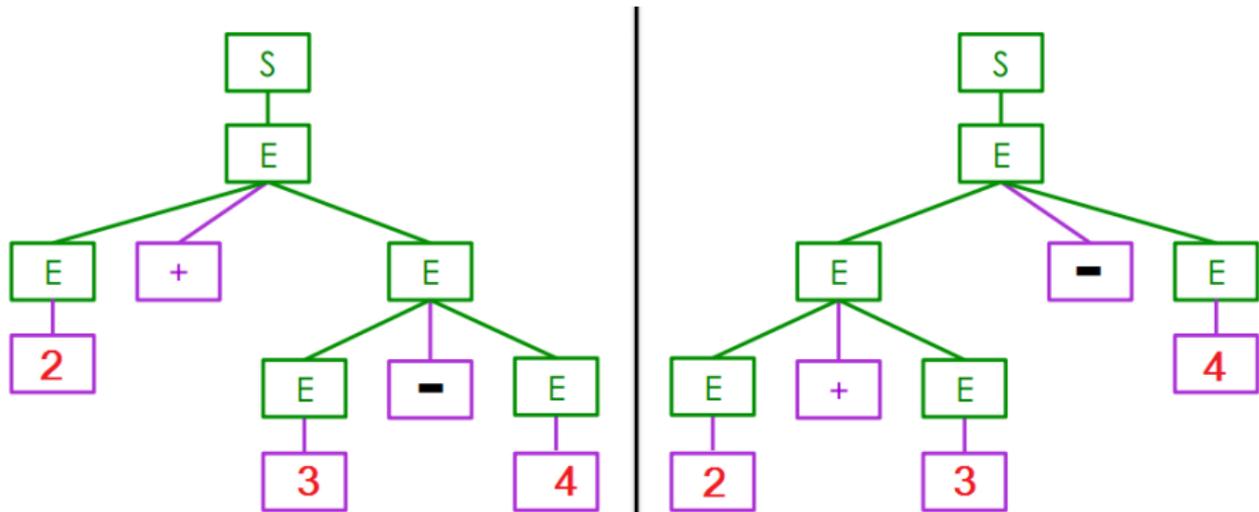
Maria disse à amiga que seu pai havia chegado.



Gramáticas Ambíguas

Geram pelo menos uma cadeia com **duas árvores diferentes**
(derivações mais à esquerda-direita)

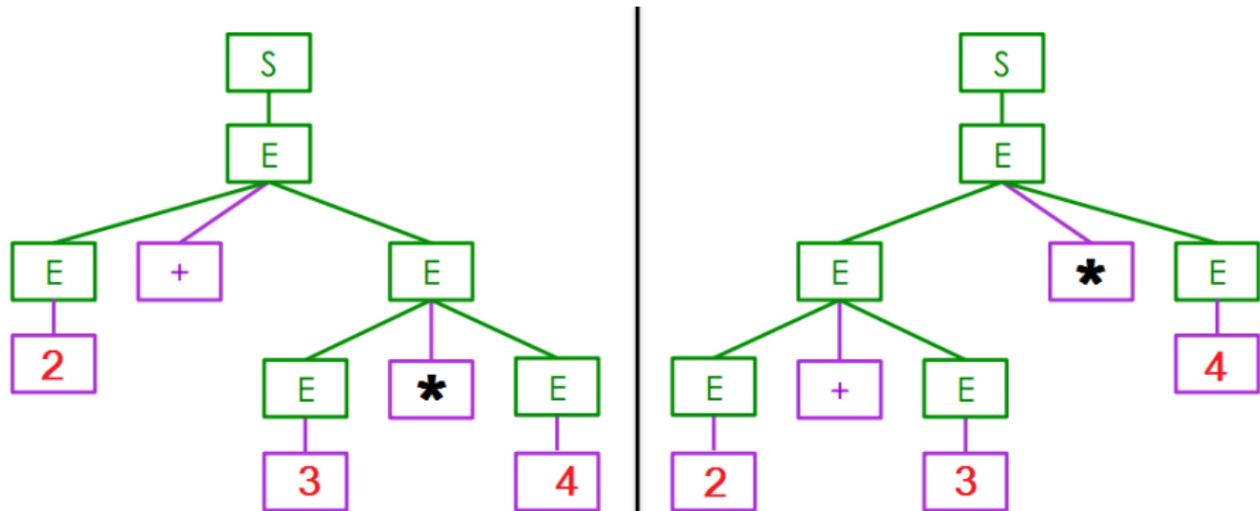
Nas linguagens de programação \Rightarrow Duas interpretações diferentes
para o mesmo programa: **programador** vs **compilador**



Gramáticas Ambíguas

Geram pelo menos uma cadeia com **duas árvores diferentes**
(derivações mais à esquerda-direita)

Nas linguagens de programação \Rightarrow Duas interpretações diferentes
para o mesmo programa: **programador** vs **compilador**





Gramáticas Ambíguas

⇒ Duas interpretações (e resultados!) diferentes para o mesmo programa: **programador** vs **compilador**

Exemplo: Dangling **else**

$$\begin{aligned} stmt \rightarrow & \mathbf{if\ expr\ then\ stmt} \mid \\ & \mathbf{if\ expr\ then\ stmt\ else\ stmt} \mid \mathbf{other} \end{aligned}$$



Gramáticas Ambíguas

⇒ Duas interpretações (e resultados!) diferentes para o mesmo programa: **programador** vs **compilador**

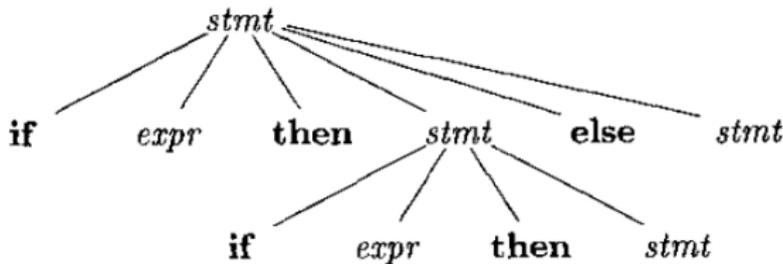
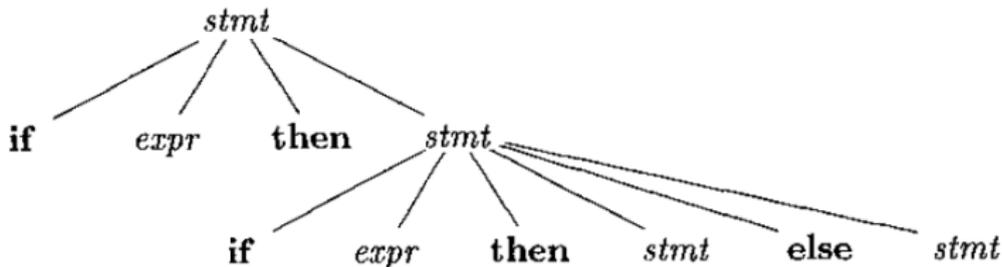
Exemplo: Dangling **else**

$$\begin{aligned} stmt \rightarrow & \mathbf{if\ expr\ then\ stmt} \mid \\ & \mathbf{if\ expr\ then\ stmt\ else\ stmt} \mid \mathbf{other} \end{aligned}$$

Seja a cadeia: **if expr then if expr then other else other**

Gramáticas Ambíguas

⇒ Duas interpretações (e resultados!) diferentes para o mesmo programa: **programador** vs **compilador**





Gramáticas Ambíguas

- ▶ A ambiguidade pode estar na gramática e não na linguagem.

Exemplo: $S \rightarrow S S \mid a$

Porém, existem linguagens inerentemente ambíguas, i.e. toda gramática que gera a linguagem é ambígua

Exemplo: $\{ a^i b^j c^k \mid i = j \text{ ou } j = k \}$



Gramáticas Ambíguas

- ▶ A ambiguidade pode estar na gramática e não na linguagem.

Exemplo: $S \rightarrow S S \mid a$

Porém, existem linguagens inerentemente ambíguas, i.e. toda gramática que gera a linguagem é ambígua

Exemplo: $\{ a^i b^j c^k \mid i = j \text{ ou } j = k \}$

- ▶ Como determinar se uma gramática é ambígua? Problema não decidível \Rightarrow não existe algoritmo



Gramáticas Ambíguas

- ▶ A ambiguidade pode estar na gramática e não na linguagem.

Exemplo: $S \rightarrow S S \mid a$

Porém, existem linguagens inerentemente ambíguas, i.e. toda gramática que gera a linguagem é ambígua

Exemplo: $\{ a^i b^j c^k \mid i = j \text{ ou } j = k \}$

- ▶ Como determinar se uma gramática é ambígua? Problema não decidível \Rightarrow não existe algoritmo

Condição suficiente: $\exists A \in N$ s.t. $A \rightarrow^* \alpha A$ e $A \rightarrow^* A \beta$



Gramáticas Ambíguas

- ▶ A ambiguidade pode estar na gramática e não na linguagem.

Exemplo: $S \rightarrow S S \mid a$

Porém, existem linguagens inerentemente ambíguas, i.e. toda gramática que gera a linguagem é ambígua

Exemplo: $\{ a^i b^j c^k \mid i = j \text{ ou } j = k \}$

- ▶ Como determinar se uma gramática é ambígua? Problema não decidível \Rightarrow não existe algoritmo

Condição suficiente: $\exists A \in N$ s.t. $A \rightarrow^* \alpha A$ e $A \rightarrow^* A \beta$

- ▶ Como eliminar a ambiguidade de uma gramática? não existe algoritmo. Tentar re-escrever a gramática incluindo aspectos semânticos. **Garantir a mesma linguagem!**



Gramáticas Ambíguas

- ▶ Eliminar a ambiguidade \Rightarrow
 1. incluir aspectos semânticos: e.g. para expressões regras de associatividade e precedência

Gramática ambígua

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + E \mid E * E \mid \\ &\quad (E) \mid \text{id} \end{aligned}$$

Gramática não ambígua

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow \text{id} \mid (E) \end{aligned}$$

2. pode gerar uma gramática maior e mais complexa.



Dangling else

Gramática ambígua

$$\begin{aligned} stmt \rightarrow & \mathbf{if\ expr\ then\ stmt} \mid \\ & \mathbf{if\ expr\ then\ stmt\ else\ stmt} \mid \mathbf{other} \end{aligned}$$

Gramática não ambígua

$$\begin{aligned} stmt \rightarrow & \mathit{matchedstmt} \mid \mathit{openstmt} \\ \mathit{matchedstmt} \rightarrow & \mathbf{if\ expr\ then\ matchedstmt\ else\ matchedstmt} \mid \\ & \mathbf{other} \\ \mathit{openstmt} \rightarrow & \mathbf{if\ expr\ then\ stmt} \mid \\ & \mathbf{if\ expr\ then\ matchedstmt\ else\ openstmt} \end{aligned}$$



Sumário

Gramáticas e Linguagens Livres de contexto

Árvore de Derivação

Gramáticas ambíguas

Análise Sintática (Parsing)

Bibliografia

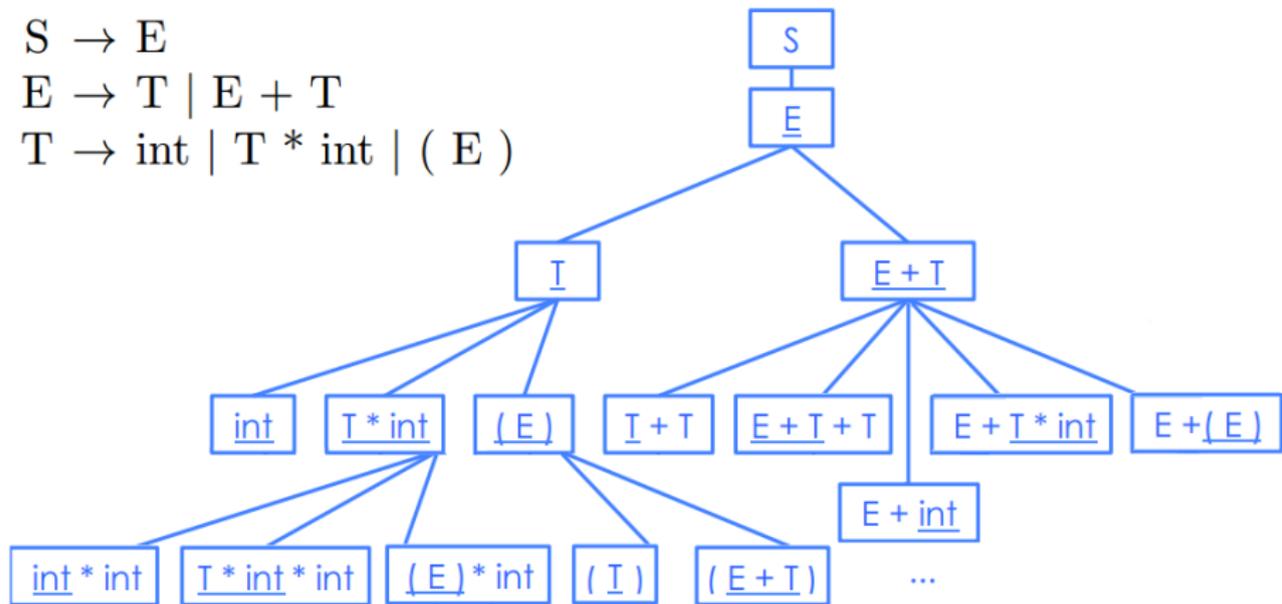
Parsing como Busca sobre Grafos

- ▶ Começar com a raiz S e gerar um nó β a partir de α se $\alpha \rightarrow \beta$

$S \rightarrow E$

$E \rightarrow T \mid E + T$

$T \rightarrow \text{int} \mid T * \text{int} \mid (E)$



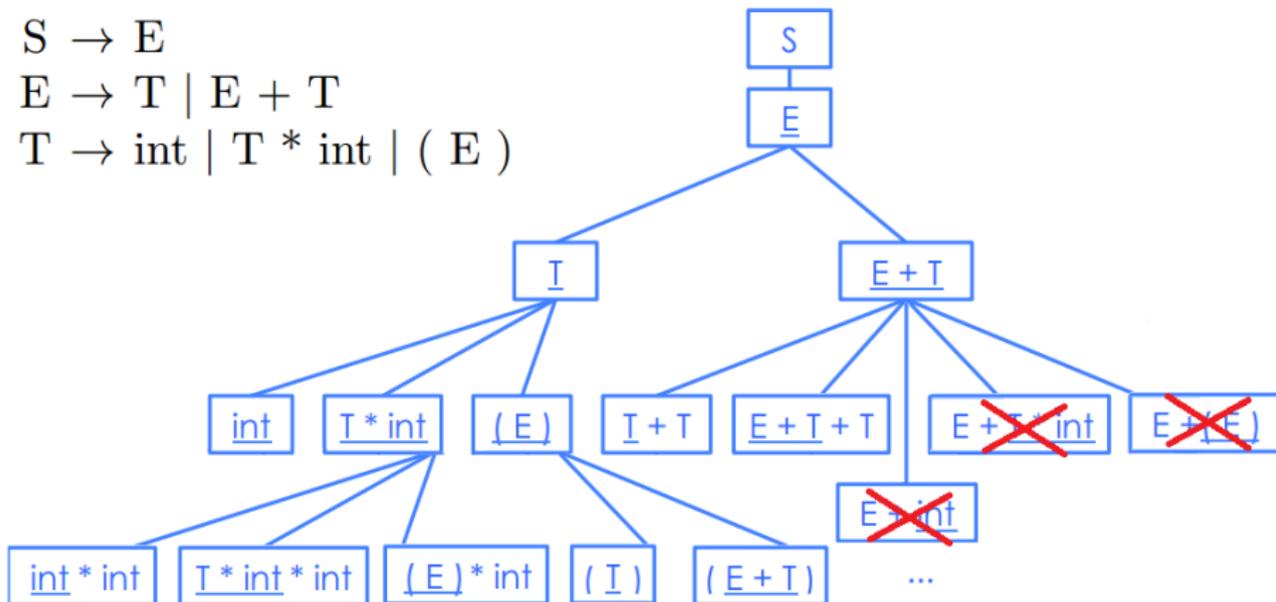
Parsing como Busca sobre Grafos

- ▶ Gerar nós que correspondam à **derivação mais à esquerda**

$$S \rightarrow E$$

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

$$T \rightarrow \text{int} \mid T * \text{int} \mid (E)$$





Parsing como Busca sobre Grafos

1. Busca em largura - mais à esquerda:

- ▶ sempre acha a derivação e pode ser adaptada para dizer se a cadeia não pertence à linguagem
- ▶ muitos ramos a partir de um nó, tempo e memória podem ser exponencial, não apropriadas para um compilador!

Exemplo: $S \rightarrow S a \mid S b \mid c$



Parsing como Busca sobre Grafos

1. Busca em largura - mais à esquerda:

- ▶ sempre acha a derivação e pode ser adaptada para dizer se a cadeia não pertence à linguagem
- ▶ muitos ramos a partir de um nó, tempo e memória podem ser exponencial, não apropriadas para um compilador!

Exemplo: $S \rightarrow S a \mid S b \mid c$

2. Busca em profundidade - mais à esquerda:

- ▶ fácil de implementar usando recursividade
- ▶ memória lineal no tamanho da derivação
- ▶ tempo exponencial no caso pior mas para muitos casos funciona rápido
- ▶ não termina se existem não terminais recursivos à esquerda - **fácil de eliminar** mas expande e modifica a estrutura da gramática



Algoritmos de Parsing

1. Busca em largura - mais à esquerda:

- ▶ sempre acha a derivação e pode ser adaptada para dizer se a cadeia não pertence à linguagem
- ▶ muitos ramos a partir de um nó, tempo e memória podem ser exponencial, não apropriadas para um compilador!

Exemplo: $S \rightarrow S a \mid S b \mid c$

2. Busca em profundidade - mais à esquerda:

- ▶ fácil de implementar usando recursividade
- ▶ memória lineal no tamanho da derivação
- ▶ tempo exponencial no caso pior mas para muitos casos funciona rápido
- ▶ não termina se existem não terminais recursivos à esquerda
- ▶ precisa de *backtracking* \Rightarrow explora várias vezes as mesmas árvores
- ▶ busca a *cegas* \Rightarrow explora árvores que não geram a cadeia fonte

Eliminação da recursividade esquerda

1. Direta: **Exemplo:** $E \rightarrow E + T \mid T$

- ▶ Transformar em recursividade direita usando um novo não terminal

$$X \rightarrow X \alpha_1 \mid \dots \mid X \alpha_n \mid \beta_1 \mid \dots \mid \beta_m$$



$$X \rightarrow \beta_1 Y \mid \dots \mid \beta_m Y$$

$$Y \rightarrow \alpha_1 Y \mid \dots \mid \alpha_n Y \mid \varepsilon$$

Eliminação da recursividade esquerda

1. Direta: **Exemplo:** $E \rightarrow E + T \mid T$

- ▶ Transformar em recursividade direita usando um novo não terminal

$$X \rightarrow X \alpha_1 \mid \dots \mid X \alpha_n \mid \beta_1 \mid \dots \mid \beta_m$$



$$X \rightarrow \beta_1 Y \mid \dots \mid \beta_m Y$$

$$Y \rightarrow \alpha_1 Y \mid \dots \mid \alpha_n Y \mid \varepsilon$$

2. Indireta: **Exemplo:** $S \rightarrow A a \mid b, A \rightarrow A c \mid S d \mid \varepsilon$

- ▶ Eliminar sistematicamente a recursividade esquerda direta ordenando os não terminais, substituindo os prefixos
- ▶ Em geral, antes é preciso eliminar ε -produções e ciclos $A \rightarrow^* A$ (ver Aho-Lam-Sethi-Ullman, capítulo 4)



Algoritmos de Parsing

A- **Busca sobre Grafos** Reconhecem qualquer LLC.

Complexidade exponencial no caso pior

1- **Busca em largura** - mais à esquerda

2- **Busca em profundidade** - mais à esquerda

- ▶ precisa de *backtracking* \Rightarrow explora várias vezes as árvores
- ▶ busca a *cegas* \Rightarrow explora árvores que não *match* a cadeia fonte

B- **Programação Dinâmica**: Armazenar sub-árvores.

Reconhecem qualquer LLC. Complexidade $O(n^3)$

3- **Algoritmo CYK (Cocke-Younger-Kasami)**: aplicável a gramáticas em formal normal de Chomsky (CNF), usa uma tabela

4- **Algoritmo de Earley**: aplicável a qualquer gramática, usa um arranjo de conjuntos de estados

Análise Sintática em Tempo Lineal

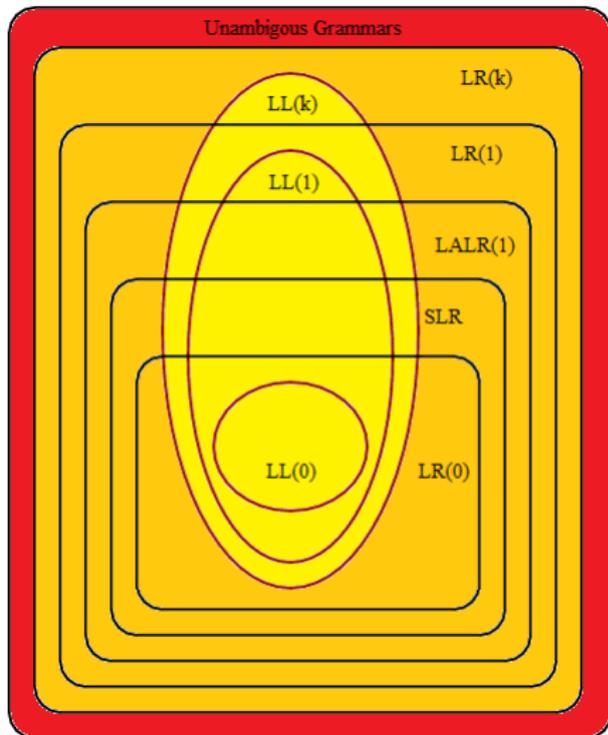
- ▶ São determinísticos:
escolhem sempre um único caminho olhando o início (k) do sufixo da cadeia fonte
- ▶ **Complexidade $O(n)$**
- ▶ Não reconhecem todas as linguagens LC
- ▶ Dois grandes tipos:

Descendentes - LL

Left-most derivation

Ascendentes - LR

Right-most derivation





Sumário

Gramáticas e Linguagens Livres de contexto

Árvore de Derivação

Gramáticas ambíguas

Análise Sintática (Parsing)

Bibliografia

Bibliografia

1. **Compilers: Principles, Techniques, and Tools** (2nd Edition). Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. Addison-Wesley, 2006
2. **Introduction to Automata Theory, Languages, and Computation** (3rd Edition). J. Hopcroft, R. Motwani and J. Ullman. Addison-Wesley, 2006
3. Introduction to the Theory of Computation. M. Sipser
4. Theory of Computation: Formal Languages, Automata, and Complexity by J. Glenn Brookshear
5. Formal Language: A Practical Introduction, A. B. Webber
6. Linguagens Formais e Autômatos, P. B. Menezes

