Teoria da Computação

Gramáticas e Linguagens Livres de contexto

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

outubro 2017

Sumário

Re-lembrando Gramáticas

Gramáticas: $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{S}, \mathbf{P})$

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 $\alpha \to \beta$ $(\alpha \to \beta_1 | \beta_2 | \ldots | \beta_n)$

Exemplo:

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

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

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

Re-lembrando Gramáticas

Gramáticas Livres de Contexto: $(\mathbf{N}, \mathbf{\Sigma}, \mathbf{S}, \mathbf{P})$

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. $\mathbf{P} \subseteq \mathbf{N} \times (\mathbf{N} \cup \mathbf{\Sigma})^*$ conjunto de **produções ou regras sintáticas** escritas $A \to \beta$ $(A \to \beta_1 | \beta_2 | \ldots | \beta_n)$

Exemplo:

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

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

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

Gramática Livre de Contexto para Expressões

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

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

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?

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

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?

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

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

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

Mais à direita

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

Arbitrária

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

Gramática Livre de Contexto para Expressões

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

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

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?

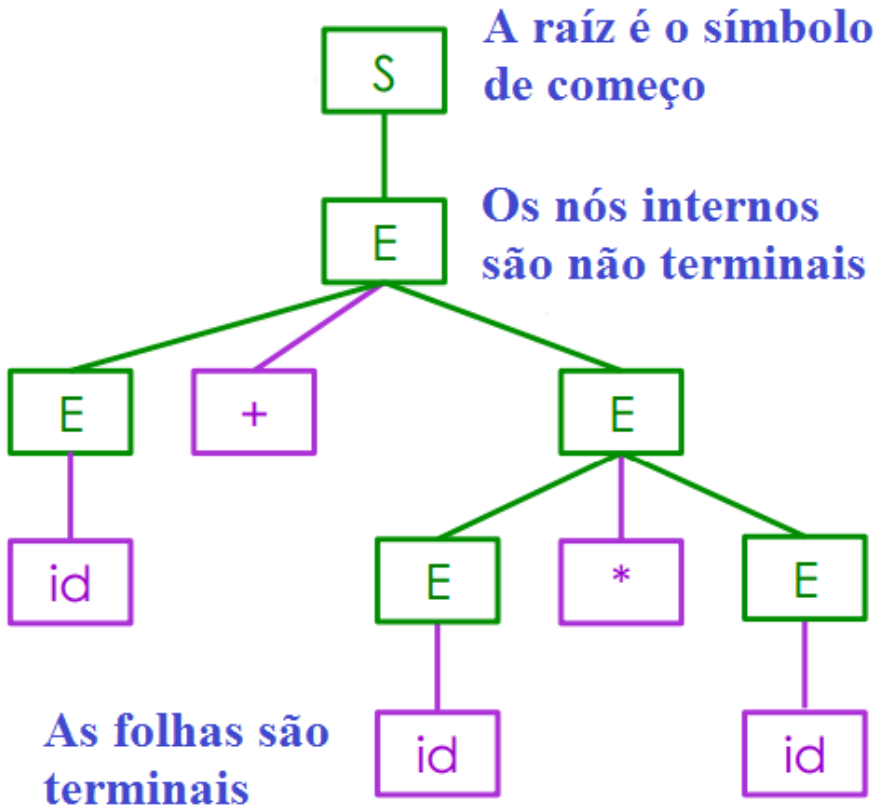
Sumário

Derivações Canônicas e Árvore de Derivação

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

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

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

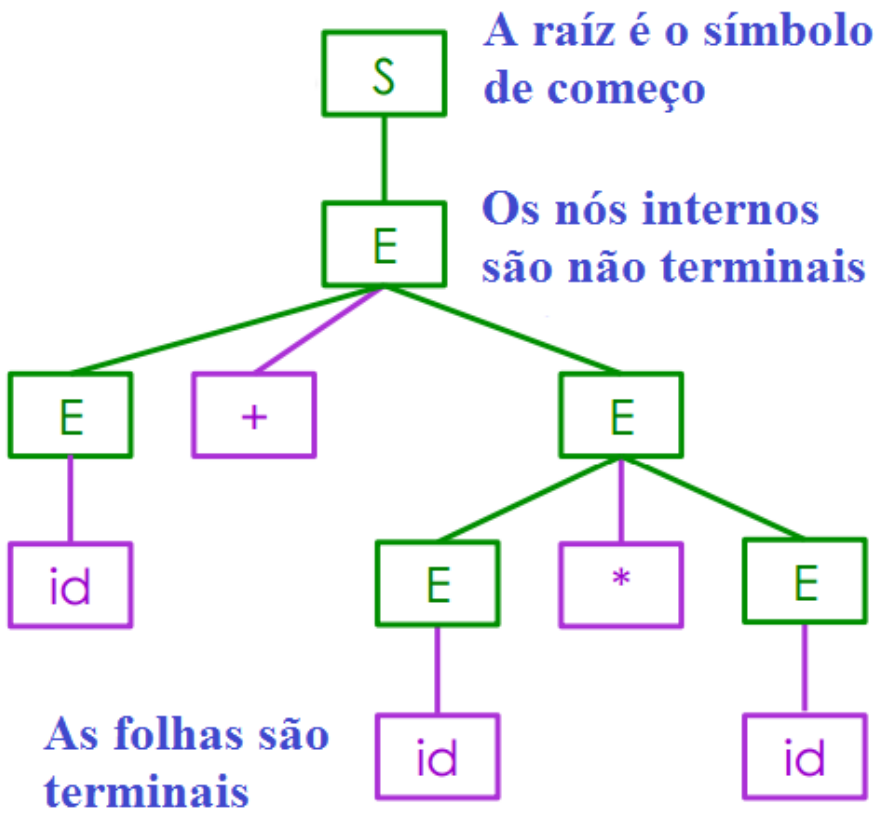


Derivações Canônicas e Árvore de Derivação

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

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

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



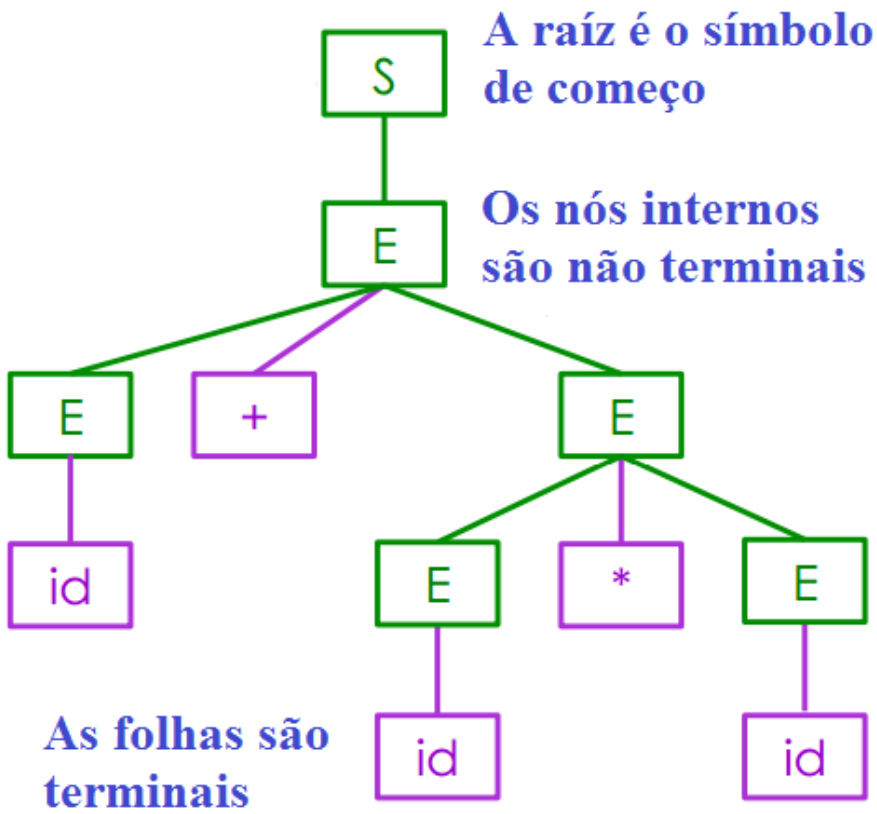
A produção aplicada gera os filhos cada nó interno da árvore

Derivações Canônicas e Árvore de Derivação

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

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

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



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

Derivações Canônicas e Árvore de Derivação

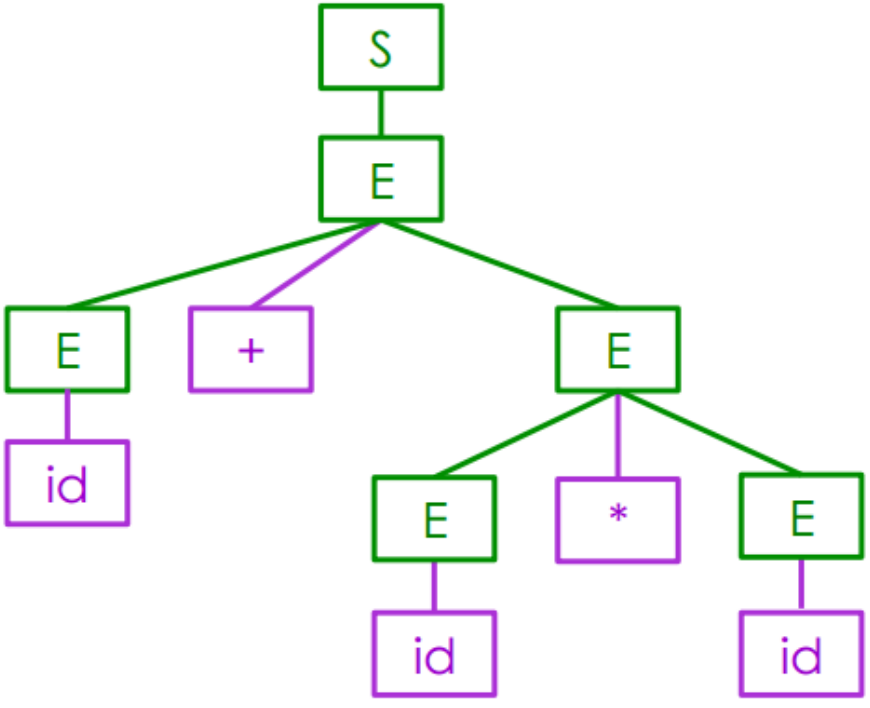
- Uma árvore de derivação representa um conjunto de derivações que todas geram a mesma cadeia
- Uma árvore de derivação representa uma única derivação mais à esquerda e uma única derivação mais à direita
- Uma gramática pode ter **duas árvores diferentes** (derivações mais à esquerda-direita) para a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

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

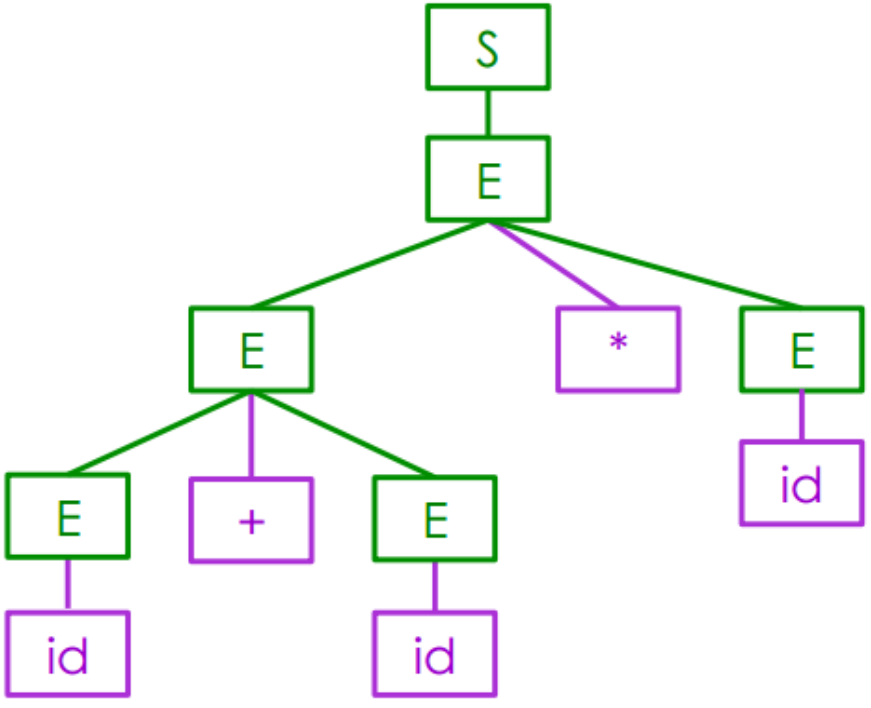
$$\mathrm{S} \rightarrow \mathrm{E}$$
$$\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{E} * \mathrm{E} \mid (\mathrm{E}) \mid \mathrm{id}$$

Derivações Canônicas, Árvore de Derivação, Ambiguidade

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



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



Sumário

Ambiguidade nas linguagens humanas

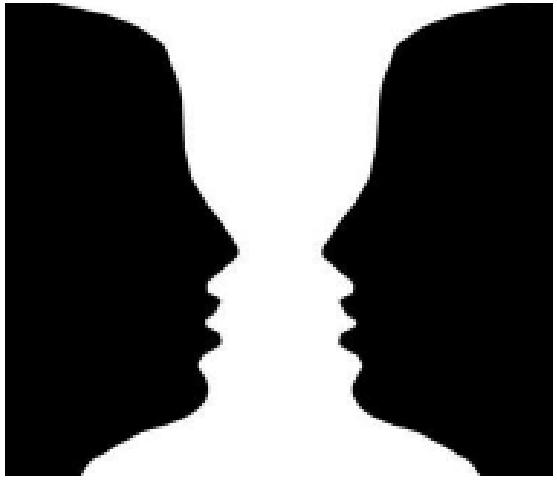
Duas interpretações diferentes para a mesma sentença/imagem

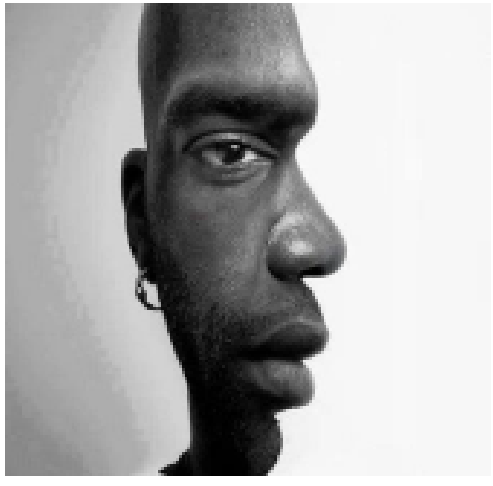
I saw the man on the hill with a telescope.

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.

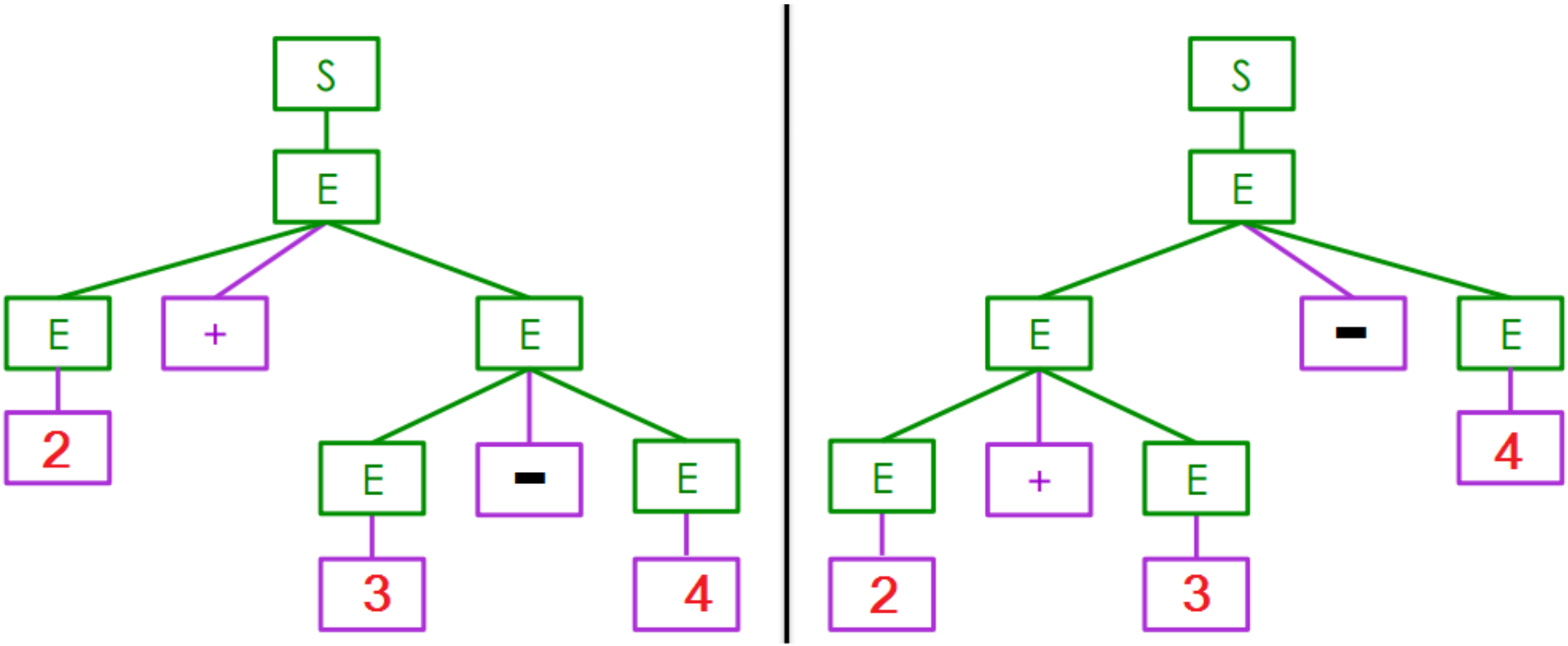




Gramáticas Ambíguas

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

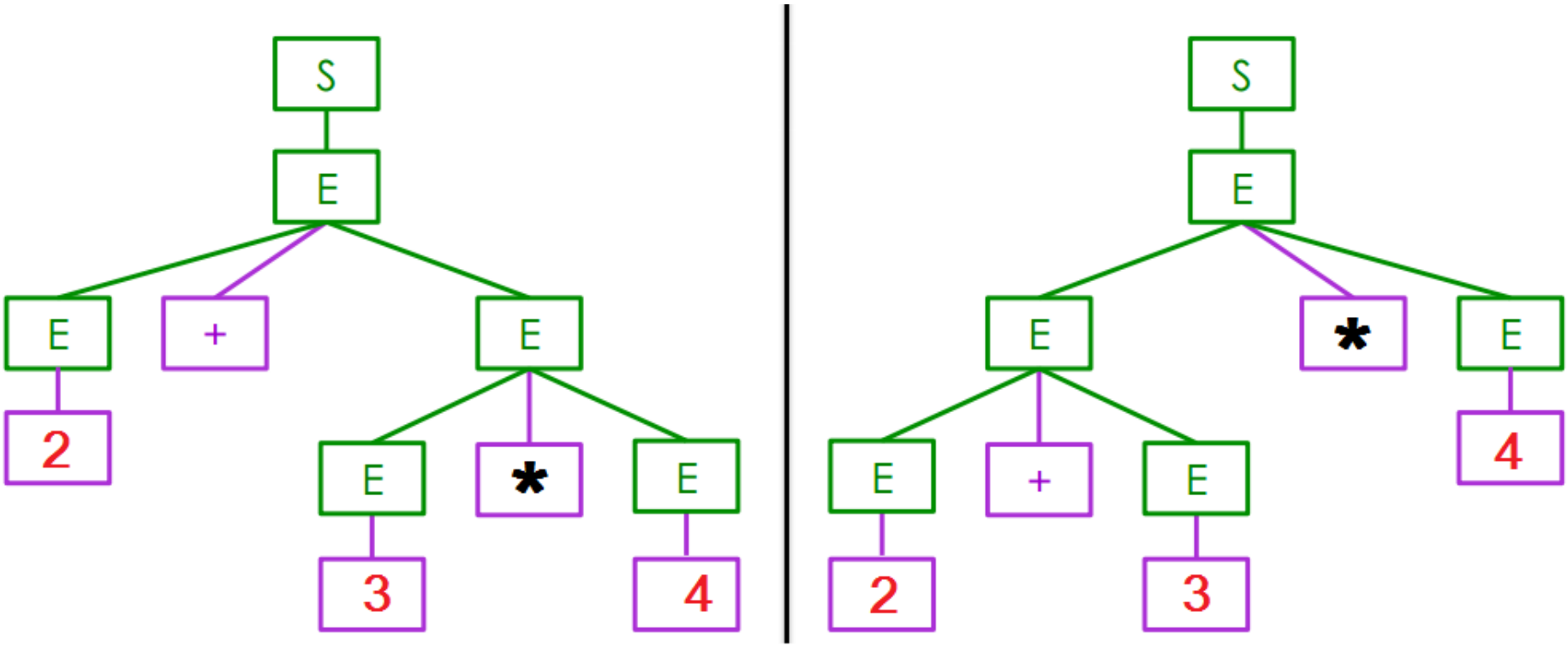
Nas linguagems de programação ⇒ Duas interpretações diferentes para o mesmo programa: **programador** vs **compilador**



Gramáticas Ambíguas

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

Nas linguagems de programação ⇒ Duas interpretações diferentes para o mesmo programa: **programador** vs **compilador**



Gramáticas Ambíguas

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

Exemplo: Dangling **else**

$$\begin{aligned} stmt \longrightarrow\ & \textbf{if}\ expr\ \textbf{then}\ stmt \mid \\ & \textbf{if}\ expr\ \textbf{then}\ stmt\ \textbf{else}\ stmt \mid \textbf{other} \end{aligned}$$

Gramáticas Ambíguas

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

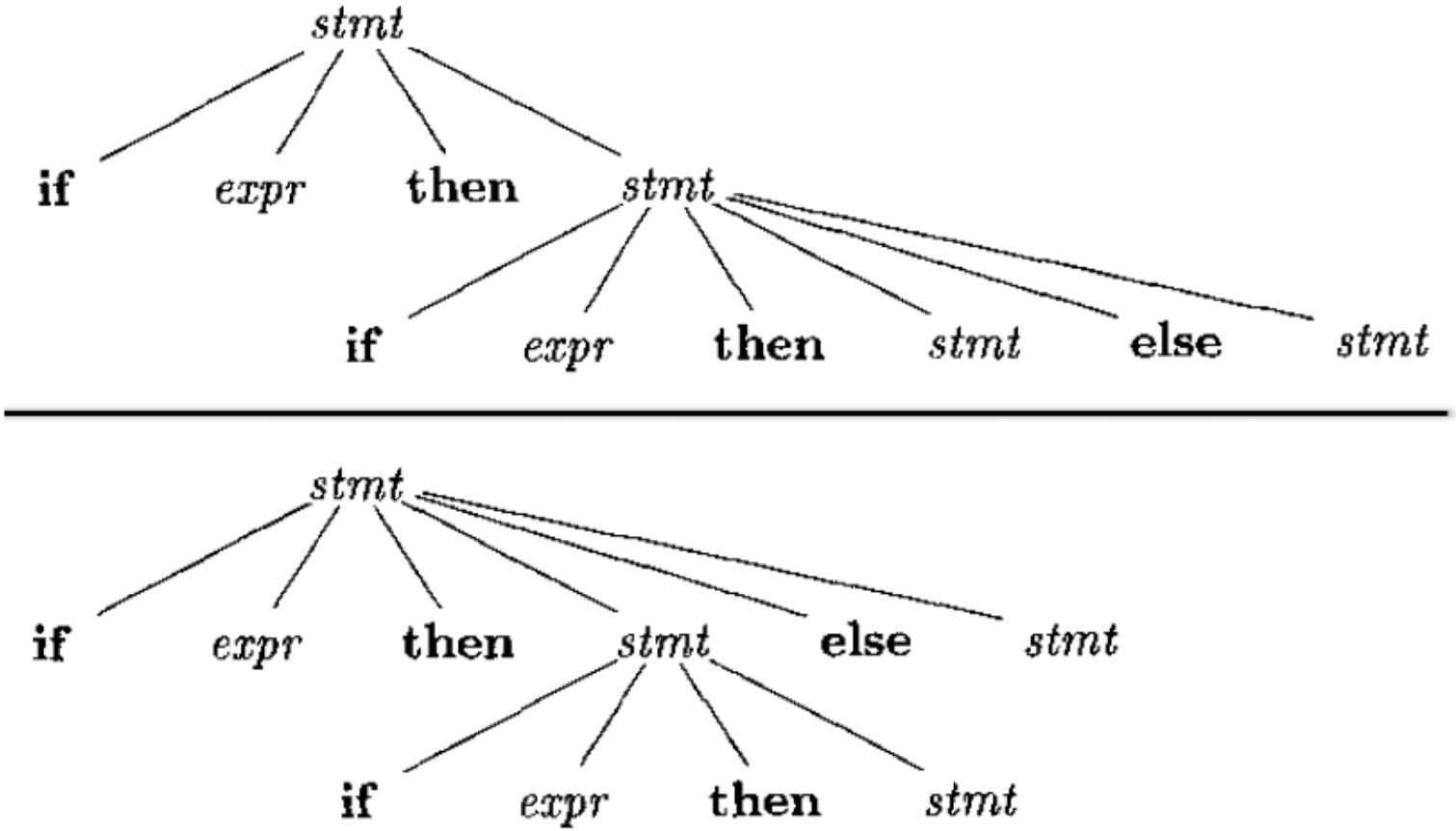
Exemplo: Dangling **else**

$$stmt \longrightarrow \textbf{if}\ expr\ \textbf{then}\ stmt \mid \textbf{if}\ expr\ \textbf{then}\ stmt\ \textbf{else}\ stmt \mid \textbf{other}$$

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

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

- 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: $\{\ \mathbf{a^i\ b^j\ c^k} \mid \mathbf{i = j}\ \ \mathbf{ou}\ \ \mathbf{j = k}\ \}$

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

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

Gramáticas Ambíguas

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

  Exemplo: $S \to S\ S \mid a$

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

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

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

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

- 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 ⇒
  1. incluir aspectos semânticos: e.g. para expressões regras de associatividade e precedência

Gramática ambígua

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

Gramática não ambígua

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

  2. pode gerar uma gramática maior e mais complexa.

Dangling else

Gramática ambígua

$$\begin{aligned} stmt \longrightarrow\ & \textbf{if}\ expr\ \textbf{then}\ stmt \mid \\ & \textbf{if}\ expr\ \textbf{then}\ stmt\ \textbf{else}\ stmt \mid \textbf{other} \end{aligned}$$

Gramática não ambígua

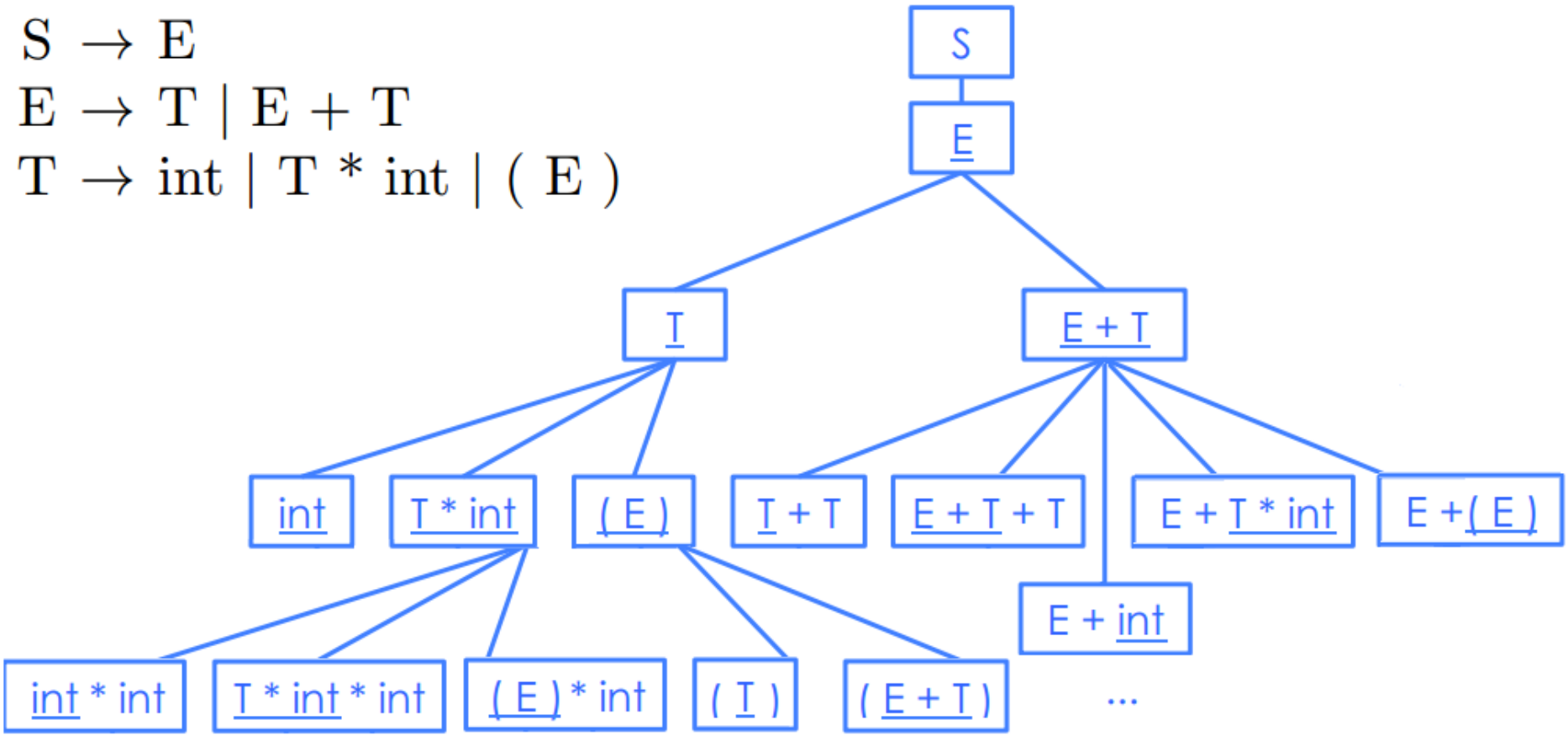
$$\begin{aligned} stmt &\longrightarrow matchedstmt \mid openstmt \\ matchedstmt &\longrightarrow \textbf{if}\ expr\ \textbf{then}\ matchedstmt\ \textbf{else}\ matchedstmt \mid \\ &\quad\ \textbf{other} \\ openstmt &\longrightarrow \textbf{if}\ expr\ \textbf{then}\ stmt \mid \\ &\quad\ \textbf{if}\ expr\ \textbf{then}\ matchedstmt\ \textbf{else}\ openstmt \end{aligned}$$

Sumário

Parsing como Busca sobre Grafos

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

$$\mathrm{S} \rightarrow \mathrm{E}$$
$$\mathrm{E} \rightarrow \mathrm{T} \mid \mathrm{E} + \mathrm{T}$$
$$\mathrm{T} \rightarrow \mathrm{int} \mid \mathrm{T} * \mathrm{int} \mid (\mathrm{E})$$

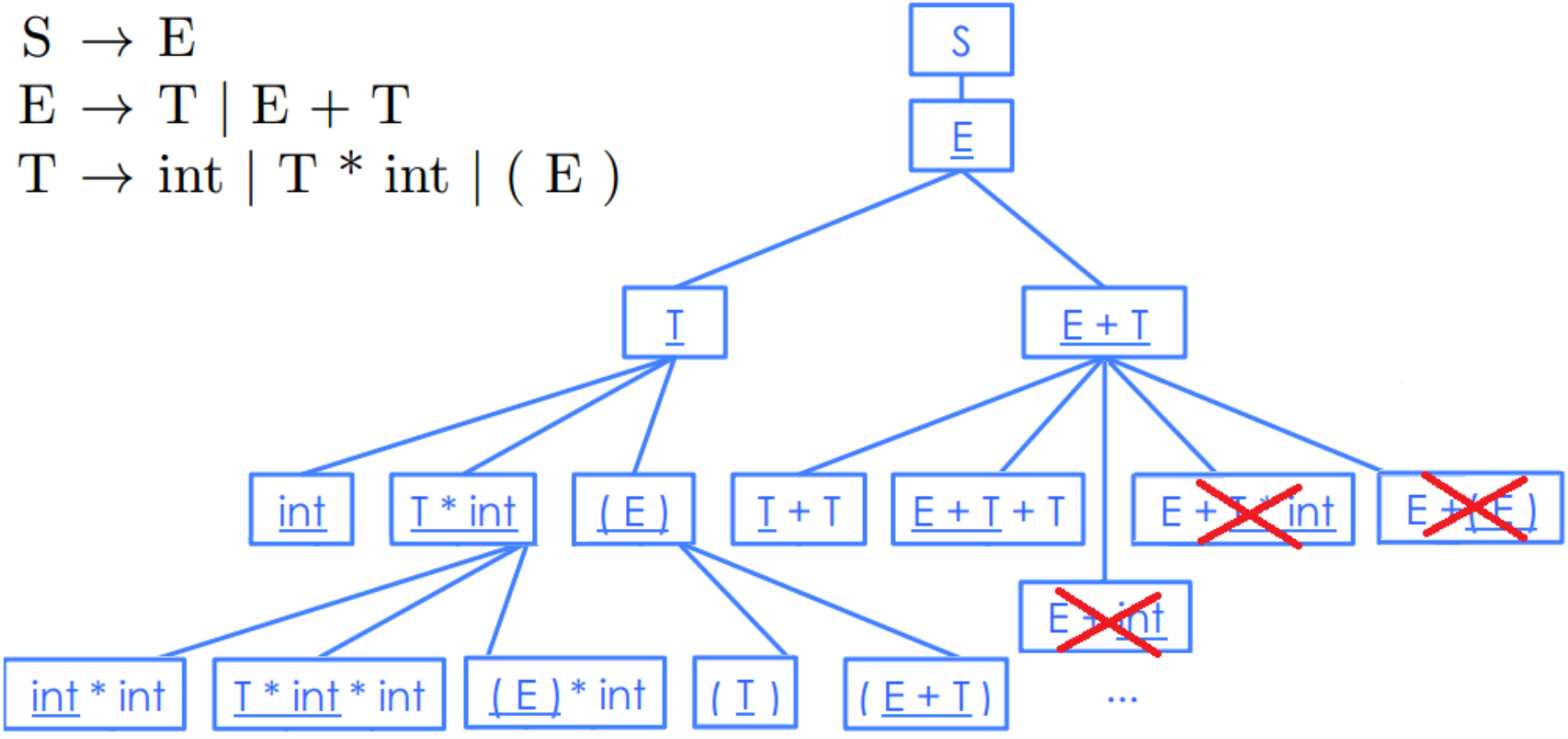


http://web.stanford.edu/class/cs143/materials/lecture05.pdf

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 memoria 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 memoria 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
   - memoria 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 memoria 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
   - memoria 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:** $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T}$
   - Transformar em recursividade direita usando um novo não terminal

$$\mathrm{X} \rightarrow \mathrm{X}\,\alpha_1 \mid \ldots \mid \mathrm{X}\,\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

$$\begin{aligned} \mathrm{X} &\rightarrow \beta_1\mathrm{Y} \mid \ldots \mid \beta_m\mathrm{Y} \\ \mathrm{Y} &\rightarrow \alpha_1\mathrm{Y} \mid \ldots \mid \alpha_n\mathrm{Y} \mid \varepsilon \end{aligned}$$

Eliminação da recursividade esquerda

1. **Direta**: **Exemplo:** $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T}$
   - Transformar em recursividade direita usando um novo não terminal

$$\mathrm{X} \rightarrow \mathrm{X}\,\alpha_1 \mid \ldots \mid \mathrm{X}\,\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

$$\begin{aligned} \mathrm{X} &\rightarrow \beta_1 \mathrm{Y} \mid \ldots \mid \beta_m \mathrm{Y} \\ \mathrm{Y} &\rightarrow \alpha_1 \mathrm{Y} \mid \ldots \mid \alpha_n \mathrm{Y} \mid \varepsilon \end{aligned}$$

2. **Indireta**: **Exemplo:** $\mathrm{S} \rightarrow \mathrm{A}\ \mathrm{a} \mid \mathrm{b}$, $\mathrm{A} \rightarrow \mathrm{A}\ \mathrm{c} \mid \mathrm{S}\ \mathrm{d} \mid \varepsilon$
   - Eliminar sistematicamente a recursividade esquerda direta ordenando os não terminais, substituindo os prefixos
   - Em geral, antes é preciso eliminar ε-produções e ciclos $\mathrm{A} \rightarrow^* \mathrm{A}$ (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 $O(n^3)$

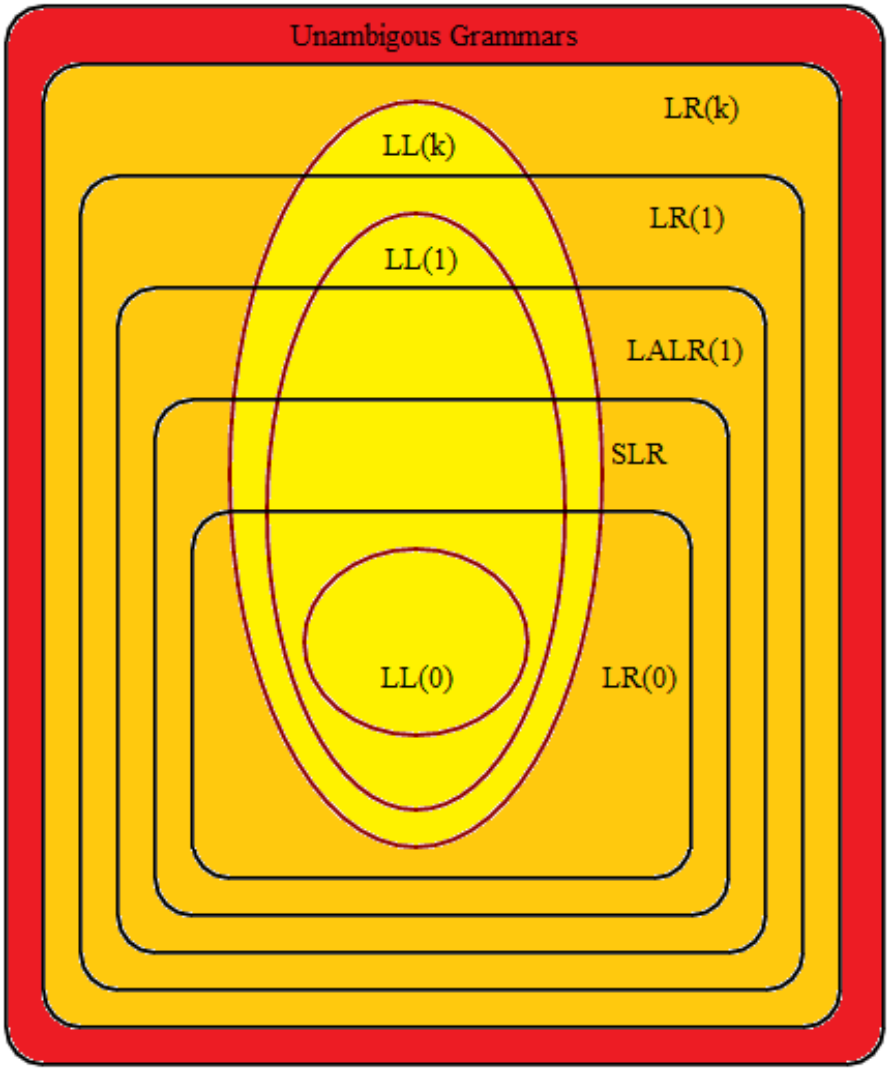
3- **Algoritmo CYK (Cocke-Younger-Kasami)**: aplicável a gramáticas em formal normal de Chomsky (CNF), usa uma tabela
4- **Algoritmo de Earley**: aplicável a qualquer gramática, usa um arranjo de conjuntos de estados

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

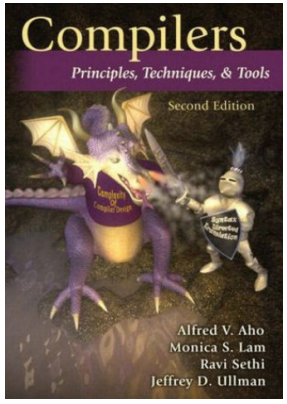
  Ascendentes - LR
  Right-most derivation



http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/12-Miscellaneous-Parsing.pdf

Sumário

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