Linguagens Formais e Autômatos

Gramáticas Livres de Contexto

Eduardo Furlan Miranda

Baseado em: GARCIA, A. de V.; HAEUSLER, E. H.
Linguagens Formais e Autômatos. Londrina: EDA, 2017.

Hierarquia de Chomsky

R	$a^n b$	ε, b, ab, aab
LC	$a^n b^n$	ab, aabb
SC	$a^n b^n c^n$	abc, aabbcc
I	a^{2^n}	a, aa, aaaa

apenas estas

Gramáticas	Regras	Ex. de linguagens geradas
GR (tipo 3) Regulares	$A \to aB$, $A \to b$, ($A \to \varepsilon$, se permitido, apenas para o símbolo inicial) $A, B \in V$ (variáveis) $a, b \in T$ (terminais)	{ ε, b, ab, aab, aaab, ... } $= \{ a^n b \mid n \geq 0 \} \cup \{\varepsilon\}$
GLC (tipo 2) Livres de Contexto	$A \to \alpha$ $A \in V$, $\alpha \in (V \cup T)^*$ A: 1 única variável	{ ab, aabb, aaabbb, aaaabbbb, ... } $= \{ a^n b^n \mid n > 0 \}$
GSC (tipo 1) Sensíveis ao Contexto	$\alpha \to \beta$ $\alpha \in (V \cup T)^+$, $\beta \in (V \cup T)^*$ $\lvert \alpha \rvert \leq \lvert \beta \rvert$ $S \to \varepsilon$, se S não aparece do lado direito de nenhuma regra	{ abc, aabbcc, aaabbbccc, ... } $= \{ a^n b^n c^n \mid n > 0 \}$
GI (tipo 0) Irrestrita ou geral	$\alpha \to \beta$ $\alpha, \beta \in (V \cup T)^*$ α: pelo menos 1 símbolo de V	{ a, aa, aaaa, aaaaaaaa, ... } $= \{ a^{2^n} \mid n \geq 0 \}$

Gramáticas Livres de Contexto (GLC)

- Todas as regras têm exatamente uma variável (e nenhum outro símbolo) do lado esquerdo

- Uma gramática $G = (V , T , P , S)$ é GLC se, e somente se, todas as regras são da forma

$$A \rightarrow \alpha \quad \text{com} \quad A \in V \quad \text{e} \quad \alpha \in (V \cup T)^*$$

- α, γ, β representam cadeia de símbolos compostas por qualquer combinação
- gramática = conjunto de regras

R	$a^n b$	ε,b,ab,aab	A→b, A→ε, A→aB
LC	$a^n b^n$	ab, aabb	A→α , α∈(V∪T)*
SC	$a^n b^n c^n$	abc, aabbcc	α→β
I	$a^{2\hat{}n}$	a, aa, aaaa	α→β

Exemplo

- Seja a linguagem que possui os parênteses corretamente balanceados
  L_{par} = { ε , () , (()) , ()() , ((())) , (())() , ()(()) , ()()() , (()()) , ... }
  - A gramática deve gerar cadeias com os caracteres "(" e ")" de forma que cada abertura de parênteses corresponde a um fechamento de parênteses posterior
    - Os parênteses devem estar corretamente balanceados

- Esta linguagem é gerada pela gramática
  - S → ε
  - S → SS
  - S → (S)

GLC

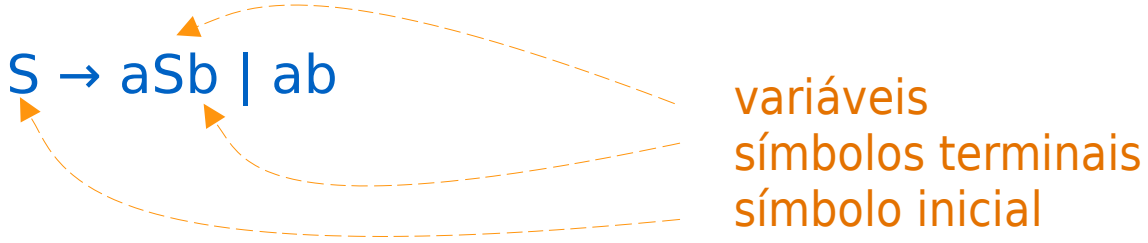
- Toda gramática regular também é uma gramática livre de contexto (GLC)
- Uma linguagem é livre de contexto (LLC) se existe uma gramática livre de contexto (GLC) que a gere
- Toda linguagem regular possui uma gramática regular que a gera, livre de contexto (LC)
  - Portanto, a linguagem L também é livre de contexto
    - A recíproca não é verdadeira

Gramáticas	Regras	Ex. de linguagens geradas
GR (tipo 3) Regulares	A→aB, A→b, (A→ε, se permitido, apenas para o símbolo inicial) A, B ∈ V (variáveis) a, b ∈ T (terminais)	{ ε, b, ab, aab, aaab, ... } = { $a^n b$ | n ≥ 0} ∪ {ε}
GLC (tipo 2) Livres de Contexto	A → α A ∈ V , α ∈ $(V \cup T)^*$ A: 1 única variável	{ ab, aabb, aaabbb, aaaabbbb, ... } = { $a^n b^n$ | n > 0 }

Exemplo

- Considere a linguagem L_p = { ab, aabb, aaabbb, ... } gerada pela gramática livre de contexto G_p :

  - S → aSb | ab



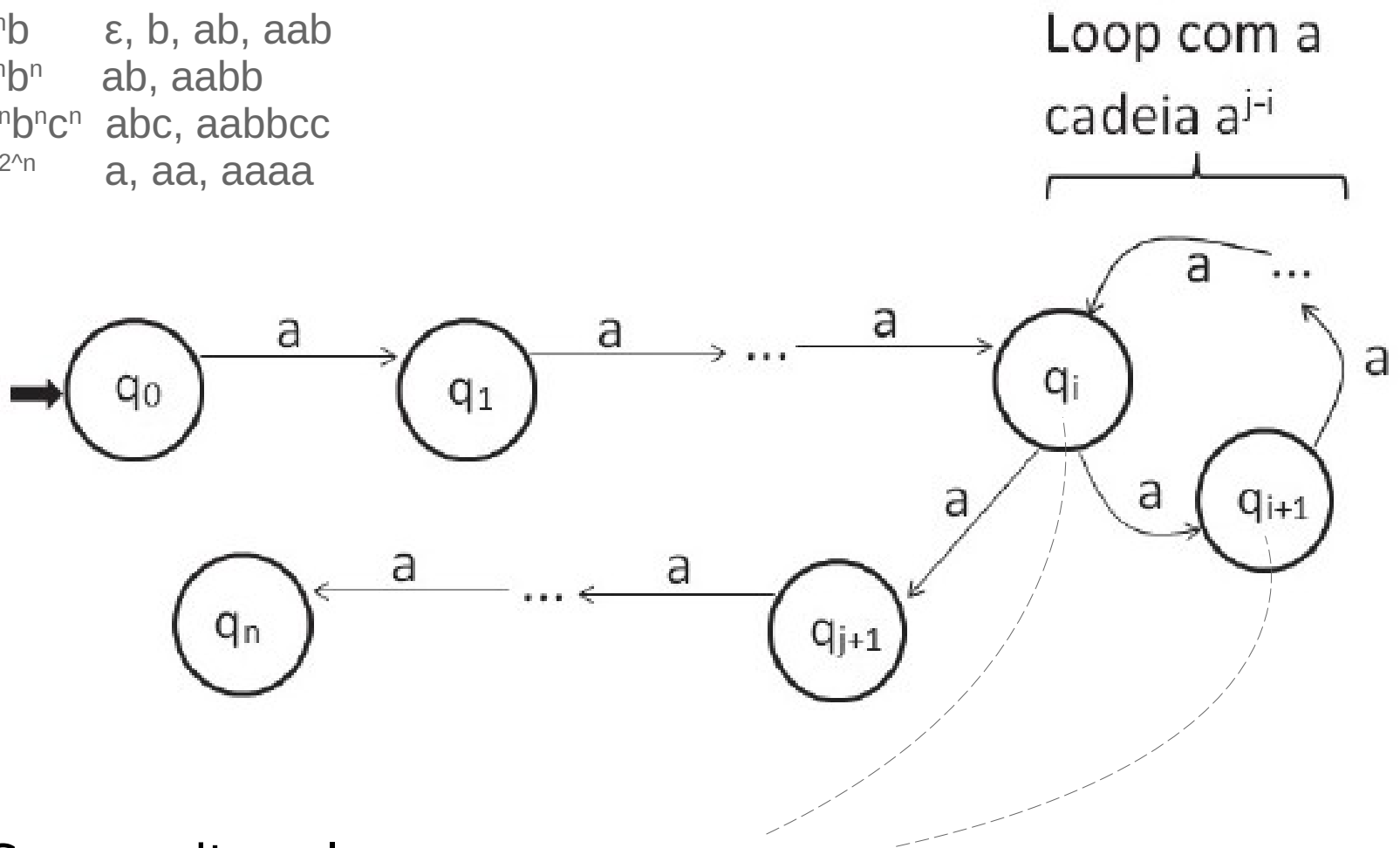
- A gramática G_p gera facilmente a cadeia aaabbb, através da derivação:
  - S ⇒ aSb ⇒ aaSbb ⇒ aaabbb

- Pode-se demonstrar que L_p não é uma linguagem regular

  Um autômato finito não pode contar a quantidade de "a"s e garantir que haja a mesma quantidade de "b"s depois. Isso exige uma memória maior do que um autômato finito pode fornecer, o que só é possível em gramáticas livres de contexto, que usam pilhas

GLC - repetição de estados

R	$a^n b$	ε, b, ab, aab
LC	$a^n b^n$	ab, aabb
SC	$a^n b^n c^n$	abc, aabbcc
I	$a^{2\wedge n}$	a, aa, aaaa



- GLCs permitem laços, como em q_i e q_{i+1}
  - Demonstra o maior poder de expressividade das GLC

GLC

- GLCs podem ser usadas para especificar a sintaxe de uma linguagem de programação (ex.: Java)
- São poderosas o suficiente para descrever estruturas sintáticas complexas, como expressões matemáticas, estruturas de controle (if, while, for) e chamadas de funções
- Ferramentas como Yacc, Bison e ANTLR utilizam GLC para gerar analisadores sintáticos (parsers) que verificam a conformidade do código-fonte com as regras da linguagem

GLC - exemplo

- Gramática livre de contexto G_{exp} que gera as expressões aritméticas formadas com as operações soma e multiplicação e o número “1”
  - S → S + S
  - S → S × S
  - S → (S)
  - S → 1

S → S+S → 1+1
S → S+S → 1+1
S → (S) → (S+S) → (1+1)

1 , 1 + 1 , 1 + 1 × 1 ,
(1 + 1) × (1 + 1) , ...

- símbolos terminais: + , × , (,) , 1
- símbolo inicial: S

A partir do símbolo inicial S, aplicamos repetidamente as regras até não haver mais variáveis

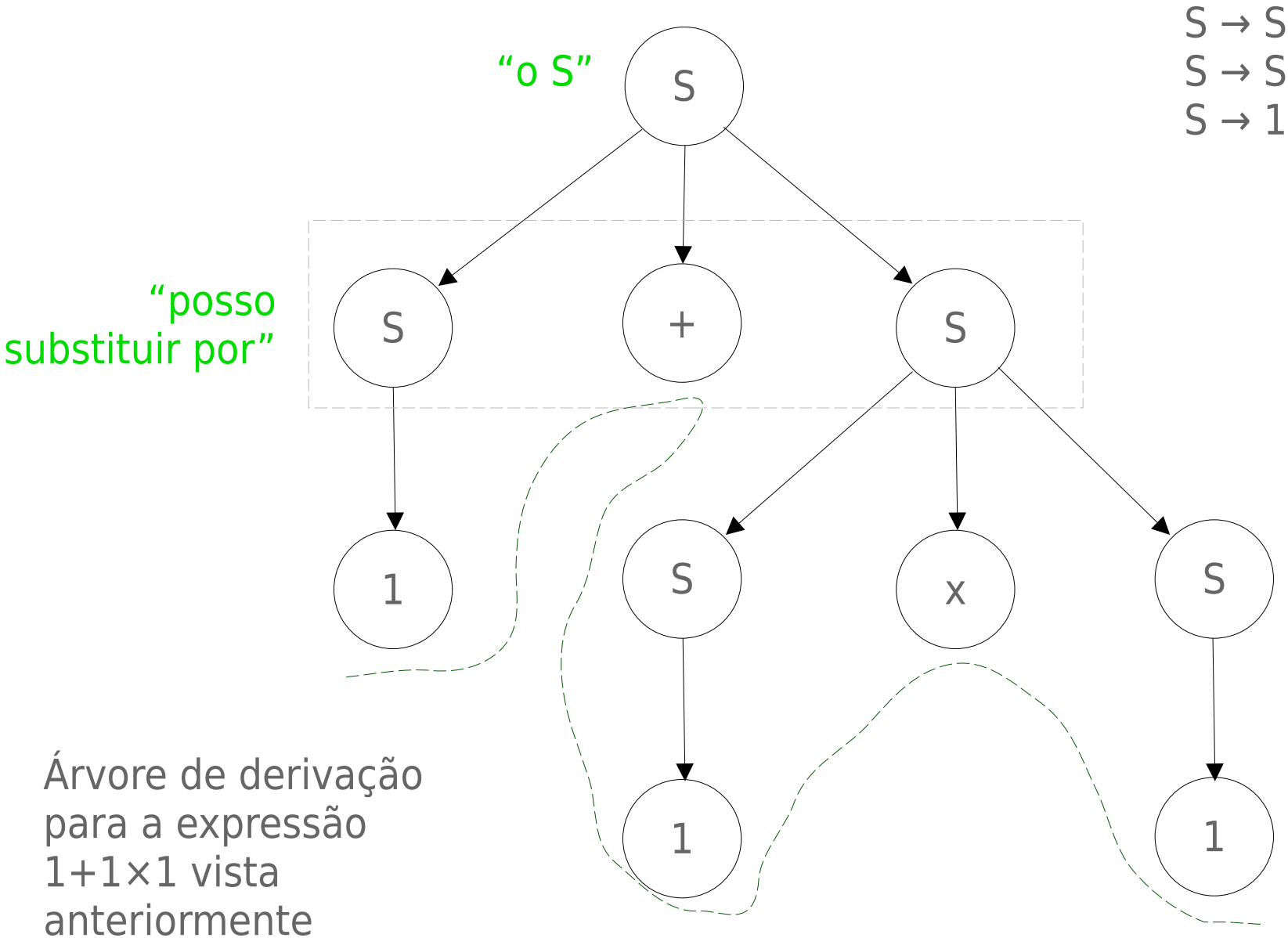
- Uma mesma cadeia pode ter diversas derivações
  - Ex.: cadeia 1 + 1 × 1 possui, entre outras, as derivações:

1) $S \Rightarrow S + S \Rightarrow 1 + S \Rightarrow 1 + S \times S \Rightarrow 1 + 1 \times S \Rightarrow 1 + 1 \times 1$
2) $S \Rightarrow S + S \Rightarrow S + S \times S \Rightarrow S + S \times 1 \Rightarrow S + 1 \times 1 \Rightarrow 1 + 1 \times 1$
3) $S \Rightarrow S + S \Rightarrow S + S \times S \Rightarrow 1 + S \times S \Rightarrow 1 + 1 \times S \Rightarrow 1 + 1 \times 1$

  - Em 1) substituímos sempre a variável mais à esquerda na forma sentencial = “Derivação Mais à Esquerda” (DME)
  - Em 2) “Derivação Mais à Direita (DMD)
  - Em 3) não há uma ordem preferencial de substituição

Árvore de derivação

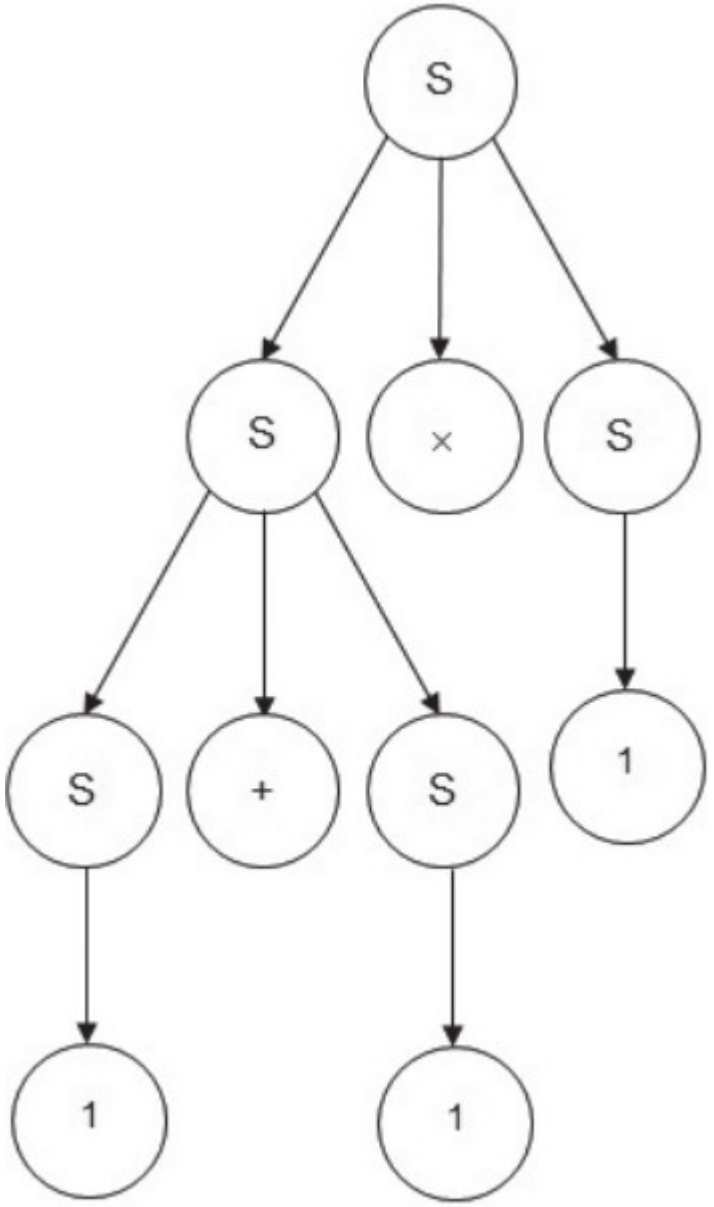
$S \to S + S$
$S \to S \times S$
$S \to 1$



Árvore de derivação para a expressão 1+1×1 vista anteriormente

Outro exemplo de árvore

- Derivações
  - S ⇒ S × S ⇒ S + S × S ⇒ 1 + S × S ⇒ 1 + 1× S ⇒ 1 + 1 × 1 (DME)
  - S ⇒ S × S ⇒ S × 1 ⇒ S + S ×1 ⇒ S + 1×1 ⇒ 1 + 1×1 (DMD)
  - S ⇒ S × S ⇒ S + S × S ⇒ 1 + S × S ⇒ 1 + 1× S ⇒ 1 + 1 × 1



- outra árvore de derivação para a mesma cadeia 1 + 1 x 1
- neste caso dizemos que a gramática é ambígua (possui mais de uma árvore de derivação)

GLC ambígua

- Uma GLC G é ambígua se existe uma cadeia w que possui mais de uma árvore de derivação, de acordo com G
- Uma vez que existe uma correspondência *um para um* entre árvores de derivação e DME, podemos dizer, de forma equivalente,
  - que uma GLC G é ambígua se existe uma cadeia w que possui mais de uma DME , de acordo com G
- O mesmo se dá com DMDs

GLC não ambígua

- Podemos ter uma GLC que gera a mesma linguagem, porém não é ambígua
- Porém não existe um algoritmo para obter uma gramática equivalente não ambígua que funcione para qualquer GLC

GLC não ambígua

- Esta GLC gera as expressões aritméticas, mas não é ambígua
  - $E \rightarrow E + T \mid T$
  - $T \rightarrow T \times F \mid F$
  - $F \rightarrow (E)$
  - $F \rightarrow 1$
- Uma derivação da sentença $1 + 1 \times 1$ é:
  - $E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow 1 + T \Rightarrow 1 + T \times F \Rightarrow$ $1 + F \times F \Rightarrow 1 + 1 \times F \Rightarrow 1 + 1 \times 1$

Ambiguidade inerente

- Dada uma GLC ambígua, nem sempre é possível construir uma gramática não ambígua equivalente
  - Considere as LLCs:
    - $L_1 = \{ a^n b^m c^m d^n \mid n, m > 0 \}$, e
      $L_2 = \{ a^n b^n c^m d^m \mid n, m > 0 \}$
  - A linguagem $L = L_1 \cup L_2$ é livre de contexto, mas, para todas as GLCs que a geram, as cadeias em $L_1 \cap L_2$ possuem mais de uma árvore de derivação
    - isso acontece porque cadeias que pertencem às duas linguagens podem ser derivadas seguindo as regras de L_1 ou de L_2,
      - o que gera múltiplas interpretações da mesma sentença
    - ou seja, todas as gramáticas que geram L são ambíguas
  - Nesse caso dizemos que a gramática L é inerentemente ambígua

Consequências da ambiguidade inerente

- Em linguagens de programação, gramáticas ambíguas podem causar dificuldades na análise sintática e interpretação do código

- Algumas linguagens livres de contexto podem ser reescritas de forma não ambígua, mas no caso da linguagem L, não existe uma gramática livre de contexto equivalente que seja não ambígua