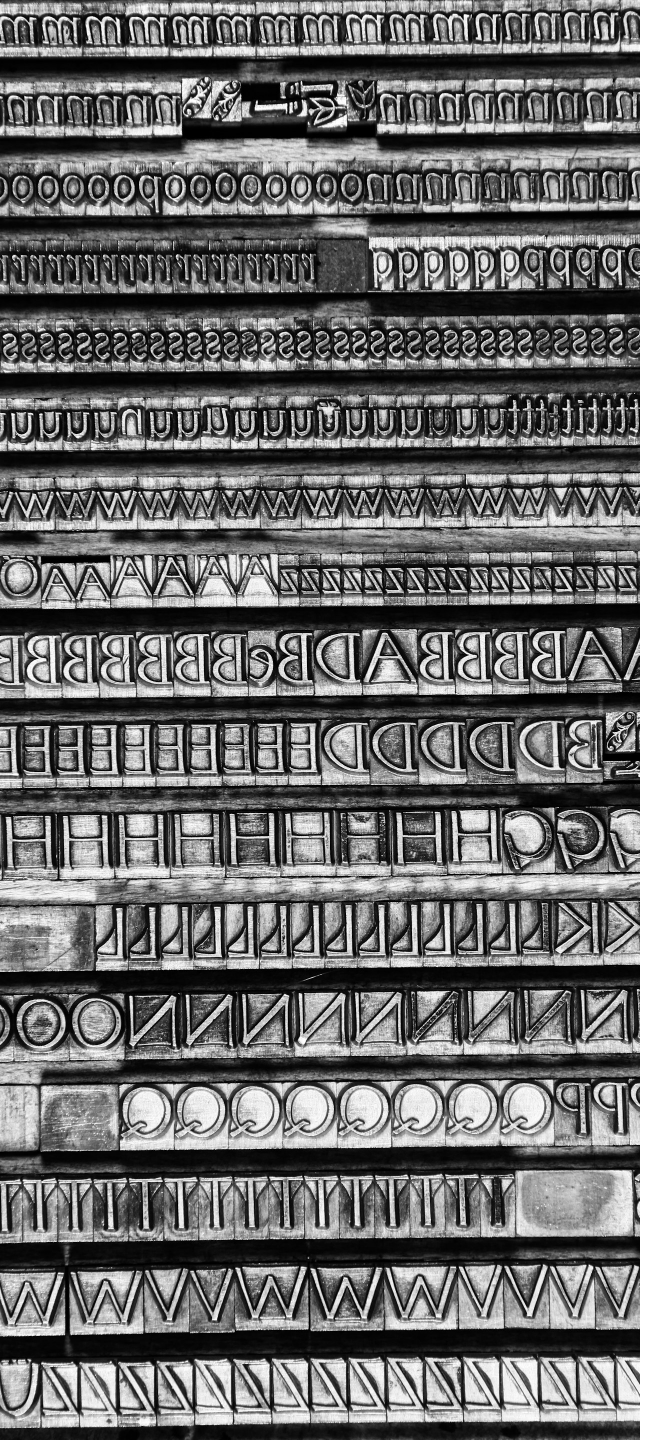

Gramáticas Livre de Contexto

Autômatos, Linguagens e Computação

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizagem

- Definir formalmente uma Gramática Livre de Contexto.
- Demonstrar o processo de derivação ($\Rightarrow^*$) de uma sentença, aplicando a sequência de regras a partir do símbolo inicial.
- Entender o cenceito de Hierarquia de Linguagens.
- Explicar a importância central das GLCs no projeto de um analisador sintático (parser) de Linguagens de Programação e
- Entender por que a utilização de Formas Normais (como a FNC) é necessária para eliminar ambiguidades e garantir que o parsing seja eficiente.

Gramática

- Formalismo para definir linguagens formais.
- Uma gramática mostra como gerar todas as palavras de uma linguagem.
- Utilizam recursão.

Hierarquia de Chomsky

Tipo	Linguagem	Gramática	Máquina de Aceitação
3	Regular	Regular	Autômato Finito
2	Livre de Contexto	Livre de Contexto	Autômato com Pilha
1	Sensível ao Contexto	Sensível ao Contexto	Autômato Linearmente Limitado
0	Rec. Enumerável	Irrestrita	Máquina de Turing

O Poder das GLCs

- Limitação das LR (Linguagens Regulares)
  - AFs possuem "memória limitada" (finita).
  - Não conseguem contar ou expressar estruturas de balanceamento, como $\{a^n b^n | n \geq 0\}$.
  - O Lema do Bombeamento para LRs comprova que $L = \{a^i b^i | i \geq 0\}$ não é regular.
- LLCs Aumentam a Expressão
  - GLCs permitem descrever estruturas recursivas balanceadas.
  - Exemplo clássico: Balanço de parênteses $(^n)^n$ ou blocos $begin^n end^n$.
  - Aplicação central: GLCs são fundamentais no projeto de um analisador sintático (parser), especificando a gramática formal de Linguagens de Programação (LPs).

Definição Formal

- GLC (Gramática Livre de Contexto) é uma quádrupla $G = (V, \Sigma, P, S)$.

Símbolo	Componente	Descrição
V	Conjunto de Variáveis	Símbolos não-terminais.
Σ	Alfabeto de Terminais	Símbolos da linguagem gerada.
P	Conjunto de Regras	Regras de produção (substituição). Formato: $\alpha \rightarrow \beta$
S	Símbolo Inicial	Variável de partida para a derivação ($S \in V$).

- Restrição Principal: V e Σ devem ser disjuntos ($\Sigma \cap V = \phi$).
- Variável inicial: Ao lado esquerdo da primeira regra, a menos que seja especificado o contrário.

Exemplo 1

Considere a gramática G_1 com as regras de produção abaixo:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

- $V = \{A, B\}$
- $S = A$
- $P = \{A \to 0A1 \mid B, B \to \#\}$
- $\Sigma = \{0, 1, \#\}$

Podemos gerar a cadeia $000\#111$ pela sequência de produções (uma derivação):

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Gramática da Linguagem C

https://www.lysator.liu.se/c/ANSI-C-grammar-y.html

Convenções e Componentes (V e Σ)

- Variáveis (V)
  - Representam categorias sintáticas (e.g., Sentença, Expressão, Verbo).
  - Geralmente representadas por letras maiúsculas (A, B, C...).
  - Símbolos **não-terminais**
  - Cada variável representa uma sub-linguagem.
- Terminais (Σ)
  - São os símbolos que formam as palavras finais (o alfabeto da linguagem).
  - Análogos ao alfabeto de entrada dos Autômatos Finitos.
  - Geralmente representados por letras minúsculas, números ou símbolos especiais (e.g., $a, b, 0, 1, +$).

Componentes e a Natureza "Livre de Contexto"

- **Regras de Produção (P)**
  - São regras de substituição no formato: $\mu \rightarrow \nu$.
  - Em GLCs, a estrutura é restrita:
    - **Lado Esquerdo (μ):** Deve ser uma única variável ($A \in V$).
    - **Lado Direito (ν):** Pode ser qualquer cadeia de variáveis e/ou terminais ($\nu \in (V \cup \Sigma)^*$).
- **Caractere "Livre de Contexto"**
  - A regra $A \rightarrow w$ significa que o não-terminal A pode ser substituído por w.
  - Essa substituição é **sempre válida**, independentemente do contexto (u e v) em que A aparece (i.e., $uAv \Rightarrow uwv$).
- **Símbolo Inicial (S)**
  - É a variável que inicia o processo de derivação de qualquer sentença da linguagem.

Aplicação de Regras e Notação

- **Derivação ($\Rightarrow$):** A aplicação de uma regra de substituição.
- **Exemplo de Regra:** $A \to 0A1$.
- **Mecânica de Substituição:**
  - Se na cadeia atual aparecer A, ele pode ser substituído por $0A1$.
  - Seja $\alpha = \alpha_{esq} A \alpha_{dir}$ e $A \to \beta$ uma regra, então $\alpha \Rightarrow \alpha_{esq} \beta \alpha_{dir}$.
- **Tipos de Geração (Derivação):**
  - $\alpha \Rightarrow \beta$: β é gerada **diretamente** a partir de α (1 passo).
  - $\alpha \Rightarrow^* \beta$: β é **derivável** a partir de α (aplicando 0 ou mais regras).
  - $\alpha \Rightarrow^+ \beta$: β é derivável a partir de α (aplicando 1 ou mais regras).

> A distinção entre $\to$ (regra/produção) e $\Rightarrow$ (aplicação da regra/derivação) é fundamental.

Exemplo 2

Seja $G = (\{a, b\}, \{S, A\}, P, S)$ com o conjunto de regras $P = \{S \to aAS \mid a, A \to SbA \mid SS \mid ba\}$.

$$
\begin{aligned}
S &\to aAS \\
S &\to \\
S &\to a \\
A &\to SbA \\
A &\to SS \\
A &\to ba
\end{aligned}
$$

Podemos gerar a cadeia $aabbaa$, pela derivação:

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$$

Derivação e Formas Sentenciais

- **Forma Sentencial:** Qualquer cadeia $\alpha \in (V \cup \Sigma)^*$ que é derivável a partir do símbolo inicial ($S \Rightarrow^* \alpha$). Pode conter variáveis e terminais.
- **Sentença:** Uma forma sentencial que contém **somente terminais** ($\alpha \in \Sigma^*$). É uma palavra válida na linguagem $L(G)$.
- **Derivação Mais à Esquerda:** Uma ordem de substituição que, a cada etapa, substitui a variável não-terminal mais à esquerda. Usada por analisadores sintáticos descendentes (Top-Down).

Exemplo de Derivação em Sequência

- GLC Clássica para Palíndromos: $G : S \to aSa|bSb|a|b|\lambda$.
- Derivação da Sentença 'baab':

Etapa	Aplicação	Forma Sentencial	Variável Substituída
1	$S \to bSb$	bSb	S
2	$S \to aSa$	$baSab$	S
3	$S \to \lambda$	$baab$	S

$$S \Rightarrow bSb \Rightarrow baSab \Rightarrow baab$$

- Conclusão: $baab$ é uma sentença de $L(G)$ porque $S \Rightarrow^* baab$.

Árvore de Derivação (Parse Tree)

- **Definição:** Uma estrutura de árvore que representa graficamente as substituições aplicadas na derivação.
- **Componentes:**
  - **Raiz:** O Símbolo Inicial (S).
  - **Nós Internos:** Variáveis ($A \in V$).
  - **Folhas:** Terminais ($a \in \Sigma$) ou λ.
  - **Ordem:** Os filhos de um nó interno A correspondem ao lado direito da produção $A \to X_1 X_2 \dots X_n$, lidos da esquerda para a direita.
- **Exemplo**: Árvore para 'baab' (usando a derivação anterior) na lousa.

Linguagem da Gramática

- **Definição Formal:** A linguagem gerada por uma GLC G é o conjunto de todas as sentenças (cadeias de terminais) que podem ser derivadas a partir do símbolo inicial S:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

- **LLCs vs. LRs:** Todas as Linguagens Regulares (Tipo 3) são também Linguagens Livres de Contexto (Tipo 2), mas o contrário não é verdadeiro.

Exemplos de Linguagens LLC

Linguagem	Descrição	GLC
$L_1 = \{a^i b^i \mid i \geq 1\}$	Cadeias com i 'a's seguidos de i 'b's.	$S \to aSb \mid ab$
$L_2 = \{w \mid w = w^R\}$	Palíndromos sobre $\{a, b\}$.	$S \to aSa \mid bSb \mid a \mid b \mid \lambda$
$L_3 = \{0^n 1^n 2^k \mid n, k \geq 0\}$	Balanço entre 0s e 1s, seguido por qualquer número de 2s.	$S \to AB$ $A \to 0A1 \mid \lambda \quad B \to 2B \mid \lambda$
L_4 (Parênteses aninhados)	Cadeias de parênteses bem formadas.	$S \to (S)S \mid \epsilon$

- **Observação:** A regra $A \to \lambda$ permite a geração da palavra vazia.

Propriedades de Fechamento de LLCs

- O conjunto das LLCs é fechado para algumas operações:
  - **União:** Se L_1 e L_2 são LLCs, então $L_1 \cup L_2$ também é LLC.
    - **Esboço de Prova:** Criar uma nova GLC com um novo símbolo inicial S' e regras $S' \to S_1 \mid S_2$ (onde S_1 e S_2 são símbolos iniciais de L_1 e L_2).
  - **Concatenação:** Se L_1 e L_2 são LLCs, então $L_1 L_2$ também é LLC.
  - **Fecho de Kleene ($*$):** Se L_1 é LLC, então L_1^* também é LLC.
- **Limitações (Não Fechamento):**
  - **Interseção:** O conjunto das LLCs **NÃO** é fechado para interseção.
    - Exemplo: $L_1 = \{a^i b^i c^j\}$ e $L_2 = \{a^j b^i c^i\}$ são LLCs, mas $L_1 \cap L_2 = \{a^i b^i c^i\}$ **não é LLC**.
  - **Complemento:** O conjunto das LLCs **NÃO** é fechado para complemento.

Limitações Intrínsecas

- O que LLCs não conseguem descrever:
  - Enquanto LLCs conseguem fazer balanceamento duplo (e.g., $a^n b^n$), não conseguem realizar balanceamento triplo.
- Exemplos de Linguagens NÃO Livres de Contexto:
  - Triplo Balanceamento: $L = \{a^n b^n c^n \mid n \geq 0\}$.
    - **Intuição:** O Autômato com Pilha (AP) tem apenas uma pilha. Ele consegue contar e balancear a's com b's. Mas, se precisar balancear c's também, a informação original (sobre a's ou b's) já foi perdida na pilha.
  - Cadeia Duplicada: $L = \{ww \mid w \in \{a, b\}^*\}$.
    - O AP não consegue armazenar uma palavra w inteira na pilha para depois compará-la com o segundo w da entrada.

Lema do Bombeamento para LLCs

- **Propósito:** Técnica usada para provar que uma linguagem **NÃO** é Livre de Contexto.
- **Ideia Central:** Se L é uma LLC infinita, qualquer palavra z suficientemente longa deve ter repetições de variáveis em sua árvore de derivação (ciclos).
- **Estrutura do Lema (Simplificada):**
  - Se L é uma LLC, existe uma constante k tal que para toda palavra $z \in L$ com $|z| \geq k$, z pode ser dividida em cinco partes: $z = uvwxy$.
  - As restrições são:
    a. $|vwx| \leq k$ (o 'bombeamento' deve ocorrer dentro de uma subárvore limitada).
    b. $|vx| \geq 1$ (pelo menos uma parte a ser bombeada não é vazia).
    c. $uv^iwx^iy \in L$ **para todo** $i \geq 0$ (a repetição das partes v e x deve manter a palavra na linguagem).
- **Diferença chave:** O Lema para LLCs permite o bombeamento de **duas** seções (v e x) simultaneamente, refletindo o poder de empilhar e desempilhar fornecido pela pilha.

Formas Normais (FN): Introdução e Pré-processamento

- **Motivação:** GLCs são flexíveis, mas essa liberdade dificulta a construção de analisadores sintáticos e o uso em algoritmos de reconhecimento. FNs impõem restrições nas regras, facilitando o design de parsers e provas.
- **Etapas de Simplificação (Pré-FN):** Para alcançar as FNs, transformações são necessárias.
  i. **Remoção de Recursividade no Símbolo Inicial:** Introduzir $S' \to S$ para garantir que S não seja recursivo.
  ii. **Eliminação de Regras λ (Nulidade):** Remover $A \to \lambda$ exceto $S \to \lambda$ (se $\lambda \in L(G)$).
  iii. **Eliminação de Regras de Cadeia (Unitárias):** Remover regras do tipo $A \to B$.
  iv. **Remoção de Símbolos Inúteis:** Remover variáveis que não geram terminais (TERM) e aquelas que não são alcançáveis (REACH).

Forma Normal de Chomsky (FNC)

- **Definição:** Uma GLC está na FNC se todas as suas regras de produção têm a seguinte forma:

$$A \to BC \quad (\text{onde } B, C \in V \setminus \{S\})$$
$$A \to a \quad (\text{onde } a \in \Sigma)$$
$$S \to \lambda \quad (\text{se } \lambda \in L(G))$$

- **Propriedade Chave:** As árvores de derivação na FNC são sempre **binárias**.
- **Esboço de Conversão (Regras Remanescentes):**
  i. **Terminais Misturados:** Substituir terminais na RHS que não estão sozinhos ($A \to aBC\ldots$) por novas variáveis ($A' \to a$).
  ii. **Cadeias Longas:** Substituir regras com mais de dois símbolos na RHS ($A \to X_1X_2X_3\ldots$) introduzindo novas variáveis temporárias, como em $A \to X_1T_1$, $T_1 \to X_2T_2$, e assim por diante, até o par final.

Forma Normal de Greibach (FNG)

- **Definição:** Uma GLC está na FNG se todas as suas regras de produção têm a seguinte forma:

$$A \to aA_1A_2 \dots A_n \quad (\text{onde } a \in \Sigma \text{ e } A_i \in V \setminus \{S\})$$

$$A \to a \quad \text{ou} \quad S \to \lambda$$

- **Propriedade Chave:** Toda derivação na FNG adiciona **exatamente um terminal** no início da cadeia a ser derivada.
- **Necessidade:** A FNG garante a **remoção da recursividade à esquerda** (direta e indireta). Recursividade à esquerda ($A \to A\mu$) causa **looping infinito** em analisadores Top-Down (Descendentes).
- **Esboço de Conversão:**

  i. Garantir que a GLC esteja simplificada e sem recursividade à esquerda.

  ii. Substituir variáveis no início da RHS (e.g., $A \to Bw$) usando as regras de B para forçar o primeiro símbolo a ser um terminal.

Algoritmos de Reconhecimento (Parsing)

- **Analisador Sintático (Parser):** Componente que verifica se uma palavra w pertence à linguagem $L(G)$.

Estratégia	Descrição	Simula Derivação	Exemplos de Parsers
Top-Down (Descendente)	Constrói a árvore da raiz (S) para as folhas (terminais).	Mais à Esquerda.	LL(k), Parser Recursivo Descendente.
Bottom-Up (Ascendente)	Constrói a árvore das folhas (terminais) para a raiz (S), reduzindo sequências.	Mais à Direita (reversa).	LR(k), CYK (Cocke-Younger-Kasami).

Analisadores Top-Down

- LL(k): "Left-to-right scan, Leftmost derivation, looking ahead k tokens."
- Mecanismo: Tenta adivinhar qual produção aplicar a uma variável (topo da pilha) olhando para os próximos k símbolos da entrada.
- Restrição: Exige que a gramática não tenha recursividade à esquerda (por isso a FNG é importante).
- Implementação (AP Descendente):
  i. Empilhar o Símbolo Inicial (S).
  ii. Se o topo for uma Variável (A): Substituir A por todas as regras $A \rightarrow w_i$. (Non-determinismo, ou escolha preditiva no LL(k)).
  iii. Se o topo for um Terminal (a): Comparar com o próximo símbolo de entrada. Se coincidir, desempilhar e consumir o símbolo de entrada.

Analisadores Bottom-Up

- LR(k): "Left-to-right scan, Rightmost derivation (in reverse), looking ahead k tokens."
- Mecanismo: A principal operação é a Redução. Quando um padrão de símbolos (terminais e/ou variáveis) corresponde ao lado direito de uma regra, ele é reduzido para a variável no lado esquerdo.
- Tipos Comuns: LR(0), SLR, LALR. Esses parsers são considerados os mais poderosos na prática (aceitam a maior classe de GLCs determinísticas).

O Autômato com Pilha e a Análise

- **AP (Autômato com Pilha):** Máquina aceitadora de LLCs (Tipo 2). Essencialmente um Autômato Finito Não-Determinístico (λ) **mais uma pilha** de memória ilimitada.
- **Função de Transição (δ):** Define a mudança de estado e o manuseio da pilha.

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\lambda\}))$$

- **Mecânica da Transição (Exemplo):** $\delta(q_i, a, A) = [q_j, B]$:
  - No estado q_i, lendo a na entrada e tendo A no topo da pilha...
  - ...vai para o estado q_j, desempilha A, e empilha B.

Aplicação em Compiladores

- **Análise Léxica (Lexing/Scanning):**
  - **Função:** Agrupar caracteres de entrada em **tokens** (e.g., identificadores, palavras-chave, operadores).
  - **Formalismo:** Linguagens Regulares (LRs), Expressões Regulares (ERs) e Autômatos Finitos.
- **Análise Sintática (Parsing/Sintaxe):**
  - **Função:** Verificar se a sequência de tokens está estruturalmente correta, seguindo as regras da Linguagem de Programação (LP).
  - **Formalismo:** Gramáticas Livres de Contexto (GLCs).

Ambiguidade e a Construção da AST

- Problema da Ambiguidade:
  - Uma GLC é ambígua se uma palavra (w) possui duas ou mais árvores de derivação (ou duas ou mais derivações mais à esquerda).
  - Em LPs, isso é indesejável, pois leva a múltiplas interpretações estruturais (e.g., precedência de operadores).
  - Exemplo: $E \rightarrow E + E \mid E \times E \mid id$ para a cadeia $id + id \times id$.
- Árvore Sintática Abstrata (AST):
  - O objetivo final do parsing é gerar uma representação hierárquica da estrutura do código.
  - A AST (Abstract Syntax Tree) é uma versão simplificada da Árvore de Derivação, eliminando detalhes sintáticos irrelevantes e focando na estrutura semântica para posterior geração de código.

Exercícios de Criação de GLCs

1. **GLC para Linguagem com Contagem Desbalanceada (LLC):**
   - Desenvolva uma GLC para $L_a = \{a^n b^m \mid m > n, \text{ e } n, m \geq 0\}$. (O número de b's é estritamente maior que o número de a's).
   - **Dica:** Gaste os a's em par com b's, e garanta que reste pelo menos um b.
2. **GLC para Expressão Aritmética (Parênteses):**
   - Desenvolva uma GLC (usando variáveis E, T, F) que aceite expressões simples com soma e multiplicação (e.g., $a \times a + a$).

Exercícios de Derivação e Ambiguidade

1. **Derivação (Sequência):**
   - Dada a GLC G:

$$S \to aSa \mid aBa$$
$$B \to bB \mid \lambda$$

   - Mostre a sequência de derivação mais à esquerda para a sentença aab^2a (i.e., $aabba$).
2. **Verificação de Ambiguidade:**
   - A GLC G a seguir, que gera cadeias com o mesmo número de 'a's e 'b's, é ambígua? Justifique.

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

   - **Dica:** Tente encontrar duas derivações mais à esquerda (ou duas árvores) para a cadeia ab.

Perguntas para Reflexão (Discussão)

1. **Poder de Expressão:** Por que as GLCs, mesmo sendo mais poderosas que as LRs, ainda não conseguem descrever linguagens como $\{a^n b^n c^n\}$ ou $\{ww\}$? O que falta no modelo do Autômato com Pilha?
2. **Ambiguidade na Prática:** Por que a ambiguidade de uma gramática é um problema **grave** no contexto de um compilador de Linguagem de Programação?
3. **Recursividade vs. Forma Normal:** Qual é a relação entre recursividade à esquerda ($A \rightarrow A\mu$) e a eficiência de um analisador sintático Top-Down? Que Forma Normal resolve este problema e por quê?

Resumo / Principais Lições

- **Formalismo:** GLCs (Tipo 2) são definidas por $G = (V, \Sigma, P, S)$, onde o lado esquerdo de cada regra é uma única variável.
- **Capacidade:** GLCs descrevem estruturas que exigem **balanceamento** e **memória infinita** (e.g., parênteses aninhados, $a^n b^n$), superando as Linguagens Regulares.
- **Aceitação:** São reconhecidas pelo **Autômato com Pilha (AP)**, que usa uma pilha para memória.
- **Parsing:** A análise sintática (parsing) usa estratégias **Top-Down (LL)** ou **Bottom-Up (LR)** para construir a árvore de derivação.
- **Restrições:** FNC e FNG impõem formatos restritos às regras para facilitar algoritmos de análise e garantir a remoção de recursão à esquerda.
- **Limitação:** LLCs não conseguem balancear três ou mais quantidades distintas (e.g., $a^n b^n c^n$).