

Gramáticas e Linguagens

Livres de Contexto

$a^n b^n$



Exemplo preliminar

- Considere a linguagem dos Palíndromos L_{pal} :
 - radar, Roma é amor, 0110, 11011, ...
 - ou seja, quando $w = w^r$
- L_{pal} é regular?
 - usando o Lema do Bombeamento: Suponha L_{pal} regular, e considere $w = 0^n 1 0^n$. Se é regular, então podemos desmembrar w em $w = xyz$, tq y consiste em um ou mais 0's do prefixo. Desse modo, xz , que também teria que estar em L_{pal} , teria menos 0's à esquerda do que à direita do 1. Assim, xz não pode ser um palíndromo. O que contradiz a hipótese. Logo, L_{pal} não é regular.

Definição Recursiva de Palíndromo

- **Base:** ε , 0 e 1 são palíndromos.
- **Indução:** Se w é um palíndromo, então $0w0$ e $1w1$ também são palíndromos.
- Nenhuma cadeia é um palíndromo de 0's e 1's, a menos que ela decorra dessa base e dessa regra de indução.

Uma gramática livre de contexto é uma notação formal para expressar tais definições recursivas de linguagens.

Regras de Definição de Palíndromos

1. $P \rightarrow \varepsilon$

2. $P \rightarrow 0$

3. $P \rightarrow 1$

4. $P \rightarrow 0P0$

5. $P \rightarrow 1P1$

Base

Indução

Uma Gramática Livre de Contexto para Palíndromos

Def. Uma Gramática Livre de Contexto (GLC ou CFG) é uma quádrupla: (V_n, V_t, P, S) , onde

1. Conjunto finito de variáveis ou categorias sintáticas ou símbolos não terminais (V_n)
2. Conjunto finito de símbolos das cadeias definidas - Alfabeto de Símbolos Terminais (V_t)
3. Conjunto finito de produções ou regras (P):

*cabeça (V_n) \rightarrow corpo ($V_n \cup V_t$)**

4. Um símbolo não terminal que representa a linguagem- símbolo inicial (S)

Assumimos $V_n \cap V_t = \emptyset$. Convencionamos que $V_n \cup V_t = V$
Cada produção P tem a forma:

$$A \rightarrow \beta \quad A \in V_n; \quad \beta \in V^*$$

$S \in V_n$ denota a principal categoria gramática de G ; é dito o símbolo inicial ou o axioma da gramática.

Ex.1: $G = (\{S, A, B\}, \{a, b\}, P, S)$

$P: \{S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b\}$

Notação/Convenções

- Letras do alfabeto latino maiúsculas $\{A, B, \dots, Z\}$: **variáveis ou não terminais**
- Letras do começo do alfabeto latino minúsculas $\{a, b, c, \dots\}$: **terminais**
- Letras do fim do alfabeto latino minúsculas $\{t, u, v, x, z\}$: **cadeias de terminais**
- Letras gregas minúsculas $\{\alpha, \beta, \gamma, \delta, \epsilon, \dots, \omega\}$: **cadeias de terminais e não terminais**

alfa, beta, gama, delta, épsilon, zeta, eta, teta, iota, kapa, lâmbda, mi, ni, xi, ômicron, pi, rô, sigma, tau, úpsilon, fi, qui, psi, ômega

Exemplo

Seja a Gramática que representa uma simplificação de expressões aritméticas de uma LP, onde só permitiremos as letras a e b e os dígitos 0 e 1. Todo identificador deve começar com a ou b, que podem ser seguidos por qualquer cadeia em $\{a,b,0,1\}^*$

metalinguagem:
OU

1. $E \rightarrow I \mid E + E \mid E * E \mid (E)$

2. $I \rightarrow a \mid b \mid Ia \mid Ib \mid IO \mid I1$

→ I = Linguagem regular:
 $(a+b)(a+b+0+1)^*$

Derivações numa Gramática

(como gerar cadeias da linguagem definida pela gramática)

- Seja $G = (V_n, V_t, P, S)$
- Seja $\alpha A \beta$, com $A \in V_n$; α e $\beta \in V^*$
- Seja $A \rightarrow \gamma$ uma produção de G .
- Então dizemos que $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$

Ou seja, um passo de derivação substitui qualquer não terminal em qualquer lugar da cadeia pelo corpo de uma de suas produções.

Extensão para derivação em zero ou mais passos \Rightarrow^*

- Base: para qualquer cadeia α de V^* ,
 $\alpha \Rightarrow^* \alpha$ (*qualquer cadeia deriva a si própria*)
- Indução: Se $\alpha \Rightarrow^* \beta$ e $\beta \Rightarrow \gamma$, então $\alpha \Rightarrow^* \gamma$.
Isto é, se α pode se tornar β por meio de zero ou mais etapas, e se mais uma etapa nos leva de β para γ , então α pode se tornar γ . Em outras palavras, $\alpha \Rightarrow^* \beta$ significa que existe uma seqüência de cadeias $\gamma_1 \gamma_2 \dots \gamma_n$ para algum $n \geq 1$ tal que:
 1. $\alpha = \gamma_1$,
 2. $\beta = \gamma_n$, e
 3. para $i=1, 2, \dots, n-1$, temos $\gamma_i \Rightarrow \gamma_{i+1}$

Derivação de $a^*(a+b00)$ a partir de E

$E \Rightarrow E^*E \Rightarrow I^*E \Rightarrow a^*E \Rightarrow a^*(E) \Rightarrow a^*(E+E) \Rightarrow a^*(I+E)$
 $\Rightarrow a^*(a+E) \Rightarrow a^*(a+I) \Rightarrow a^*(a+I0) \Rightarrow a^*(a+I00) \Rightarrow$
 $a^*(a+b00)$

$\therefore E \Rightarrow^* a^*(a+b00)$

Derivações mais à esq ou mais à dir

Temos uma derivação mais à esquerda \Rightarrow_{lm} (direita \Rightarrow_{rm}) quando, a cada passo no processo de derivação de uma sentença, escolhemos derivar a variável mais à esquerda (direita)

Qualquer derivação tem uma derivação equivalente mais à esquerda e uma derivação equivalente mais à direita, i.e., $A \Rightarrow^* w$ sse $A \Rightarrow_{lm}^* w$, e $A \Rightarrow^* w$ sse $A \Rightarrow_{rm}^* w$.

A Linguagem de uma Gramática

Def. A Linguagem L gerada por uma gramática G é definida como o conjunto de cadeias terminais que têm derivações desde o símbolo inicial. Ou seja,

$$L(G) = \{x \mid x \in Vt^* \text{ e } S \Rightarrow_G^* x\}$$

Def. $L(G)$ é dita uma Linguagem Livre de Contexto - LLC (ou CFL do inglês)

Provando a Linguagem dos Palíndromos

Teo.: $L(G_{\text{pal}})$, onde G_{pal} é a gramática dos palíndromos, é o conjunto de palíndromos sobre $\{0,1\}$.

PROVA: provaremos que uma cadeia w de $\{0,1\}^*$ está em $L(G_{\text{pal}})$ **se e só se** w é um palíndromo, i.e, $w = w^r$

(Se) Suponha que w seja uma palíndromo.

Mostraremos **por indução** sobre $|w|$ que w está em $L(G_{\text{pal}})$.

Base: Se $|w|=0$ ou $|w|=1$, então w é ε , 0 ou 1. As produções de base (1,2,3) permitem concluir que $P \Rightarrow^* w$ em qualquer desses casos básicos.

Indução: Suponha verdade para $|w| \geq 2$. Tendo em vista que $w = w^r$, então $w = 0x0$ ou $w = 1x1$. Além disso, x deve ser um palíndromo, $x = x^r$.

Se $w = 0x0$, pela hipótese indutiva, afirmamos que $P \Rightarrow^* x$, ou seja, $P \Rightarrow^* 0P0 \Rightarrow^* 0x0 = w$. Se $w = 1x1$, o argumento é o mesmo, mas usamos a produção $P \rightarrow 1P1$. Em qualquer caso, concluímos que w está em $L(G_{\text{pal}})$.

(Somente se) Supomos que $w \in L(G_{\text{pal}})$ e mostramos que w é um palíndromo (por indução sobre o número de passos de derivação).

Base: Em um passo, chegamos a ε , 0 e 1 - que são palíndromos - pelas produções 1, 2 e 3.

Indução: Suponha que haja $(n+1)$ passos, $n \geq 1$, e que a hipótese seja verdadeira para todas as derivações de n passos, ou seja, se $P \Rightarrow^* x$ em n passos, então x é palíndromo.

Se há uma derivação de w em $(n+1)$ passos, então ela deve ter uma das formas:

$P \rightarrow OPO \Rightarrow^* OxO = w$ ou

$P \rightarrow 1P1 \Rightarrow^* 1x1 = w$

Em qualquer dos casos, $P \Rightarrow^* x$ em n passos.

Pela hipótese indutiva, sabemos que x é um palíndromo. Então OxO e $1x1$ também são palíndromos, o que completa a prova.

Formas Sentenciais

Todas as derivações a partir do símbolo inicial - e não apenas as cadeias terminais - possuem um papel especial. São chamadas formas sentenciais.

Def. Se $G = (V_n, V_t, P, S)$ é uma GLC, então qualquer cadeia α de $(V_n \cup V_t)^*$ tal que $S \Rightarrow^* \alpha$ é uma forma sentencial.

Se $S \Rightarrow^* l_m \alpha$, então α é uma forma sentencial à esquerda e, se $S \Rightarrow^* r_m \alpha$, então α é uma forma sentencial à direita.

Uma forma sentencial, α , é uma **sentença** de G se $\alpha \in V_t^*$ (composta apenas de terminais).

Ou seja, a Linguagem geradas pela gramática é composta pelas sentenças de G .

Exemplo de GLC

$G_1 = (\{S\}, \{0,1\}, P_1, S)$

$P_1: \{$
1. $S \rightarrow OS1$
2. $S \rightarrow O1$
 $\}$

Qual é a linguagem gerada por G_1 ? Aplicamos o processo de derivação para descobrir $L(G_1)$.

G_1

- A menor cadeia gerada é 01: $S \rightarrow 01$
- Se aplicarmos $n-1$ vezes a produção 1, seguida da produção 2 teremos:
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0^3S1^3 \Rightarrow^*$
- $0^{n-1}S1^{n-1} \Rightarrow 0^n1^n$
- Portanto, $L(G_1) = \{0^n1^n \mid n \geq 1\}$
ou $S \Rightarrow^* 0^n1^n (n \geq 1)$

Árvores de Derivação Sintática

- Em uma gramática é possível haver várias derivações equivalentes (usam as mesmas produções nos mesmos lugares MAS em ordem diferente).
- Para GLC temos uma representação gráfica que representa uma classe de equivalências chamada **Árvore de Derivação**.
- Uma **árvore de derivação** para uma GLC $G = (V_n, V_t, P, S)$ é uma **árvore rotulada ordenada** em que cada nó é rotulado por um símbolo de $V_n \cup V_t \cup \varepsilon$.
- Se um nó interno é rotulado com A e seus descendentes diretos são rotulados com X_1, X_2, \dots, X_n então $A \Rightarrow X_1X_2\dots X_n$ é uma produção de P .

Uma árvore rotulada ordenada D é uma árvore de derivação para uma GLC $G(A) = (V_n, V_t, P, A)$ se

- (1) A raiz de D é rotulada com A
- (2) Se D_1, \dots, D_k são as subárvores de descendentes diretos da raiz e a raiz de D_i é rotulada com X_i , então $A \Rightarrow X_1, \dots, X_k$ é uma produção em P .
 D_i deve ser a árvore de derivação para $G(X_i) = (V_n, V_t, P, X_i)$ se X_i é não-terminal, e D_i é um nó simples rotulado com X_i se X_i é terminal.
- (3) Se D_1 é a única subárvore da raiz D e a raiz de D_1 é rotulada com λ então $A \rightarrow \varepsilon$ é uma produção de P .

Vértices internos são não-terminais e vértices folhas podem ser não-terminais, terminais ou ε .

Quando fazemos a árvore de derivação de uma sentença, os vértices folhas são sempre terminais ou ε .

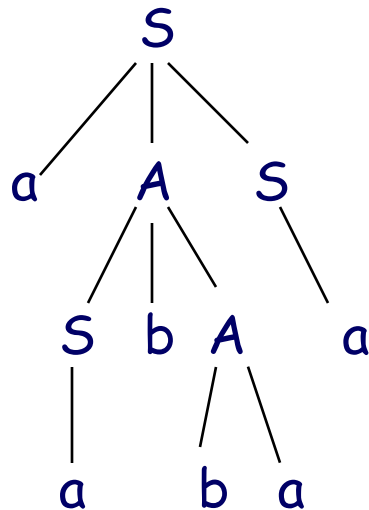
Uma sentença está representada na árvore de derivação fazendo-se a leitura das folhas (nós sem descendentes) da esquerda para direita.

Exemplos

- Árvore de derivação para a sentença $aabbaa$ da $GLC G = (\{S,A\},\{a,b\},P,S)$

P: $S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid ba \mid SS$



• Seja $G_a = (\{E, T, F\}, \{+, *, (,), a\}, P, E)$

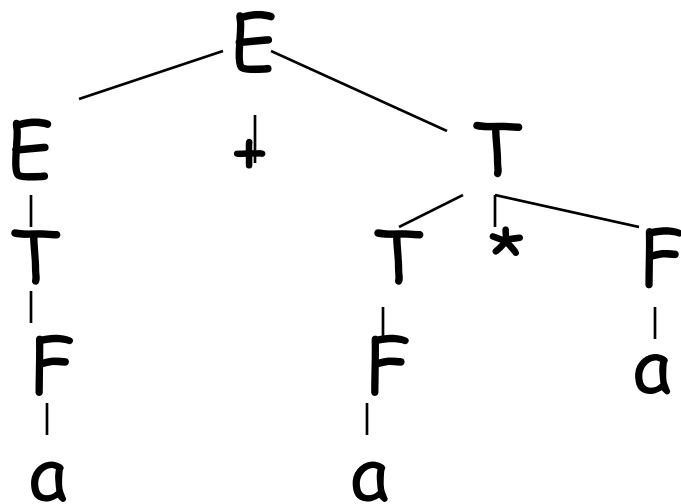
P: $E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a$

1) Quantas derivações equivalentes da sentença "a + a * a" a árvore de derivação abaixo representa?

2) Mostre a derivação *mais à esq* e a *mais à dir*.



Única Árvore de
Derivação

- Derivação mais à esquerda:

$$\begin{aligned} E &\Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T^*F \Rightarrow a+F^*F \\ &\Rightarrow a+a^*F \Rightarrow a+a^*a \end{aligned}$$

- Derivação mais à direita:

$$\begin{aligned} E &\Rightarrow E+T \Rightarrow E+T^*F \Rightarrow E+T^*a \Rightarrow E+F^*a \Rightarrow E+a^*a \\ &\Rightarrow T+a^*a \Rightarrow F+a^*a \Rightarrow a+a^*a \end{aligned}$$

De fato:

- G_b indica que o operador $*$ tem **precedência** sobre o operador $+$
- Por G_a , ambos operadores têm igual precedência.
- A linguagem de uma gramática ambígua é dita ambígua.
- A ambiguidade decorre do fato de que as árvores de derivação (no contexto de uma linguagem de programação) implicam interpretações distintas.

Ambiguidade e Derivações mais a Esquerda

Teorema: Para cada gramática

$G=(V_n, V_t, P, S)$ e cadeias w em V_t^* , w tem duas árvores de análise sintática distintas (*logo, G é ambígua*) se e somente se w tem duas derivações mais à esquerda a partir de S .

Ambigüidade nas GLC

- Um requisito importante de uma LP é que ela **não seja ambígua**.
- O mais famoso caso de ambigüidade é o **else pendente**, presente na especificação de muitas LP.

- Seja a gramática:

$C \rightarrow \text{if } b \text{ then } C \text{ else } C$

$C \rightarrow \text{if } b \text{ then } C$

$C \rightarrow s$

Ela é ambígua desde que a cadeia

$\text{If } b \text{ then if } b \text{ then } s \text{ else } s$

Pode ser interpretada como

(i) $\text{If } b \text{ then (if } b \text{ then } s \text{ else } s)$

Ou

(ii) $\text{If } b \text{ then (if } b \text{ then } s) \text{ else } s$

A primeira é a preferida em LP pois utiliza a regra informal "case o else com o if mais próximo" que resolve a ambiguidade

- Para eliminar a ambiguidade da gramática anterior, podemos reescreve-la com 2 não-terminais $C1$ e $C2$:

$C1 \rightarrow \text{if } b \text{ then } C1 \mid \text{if } b \text{ then } C2 \text{ else } C1 \mid s$

$C2 \rightarrow \text{if } b \text{ then } C2 \text{ else } C2 \mid s$

O fato de que somente $C2$ precede o `else` garante que, entre o par `then-else` gerado por qq uma das produções, deve aparecer ou um `s` ou outro `else`. Assim, a interpretação (ii) nunca ocorre.

(ii) `If b then (if b then s) else s`

Como mostrar que uma gramática é ambígua?

- Com árvores de derivação.
- **Def1:** Uma $GLC (G)$ é ambígua se há pelo menos uma cadeia pertencente à $L(G)$ com mais de uma árvore de derivação para representá-la.
- **Def2:** A existência de uma sentença com duas ou mais derivações mais à esq (ou mais à dir) caracterizam uma linguagem ambígua.

Gramática de Operadores

- Para retirar a ambiguidade:
 - introduzimos várias variáveis e estratificamos as regras, quando temos operadores com várias prioridades de resolução.
 - Para resolver a ambiguidade vinda do uso de vários operadores idênticos, forçamos o uso da recursão para esquerda ou direita (associatividade).

Regras de Precedência, Prioridade e Associatividade

- Ajudam na decisão da interpretação correta de expressões nas LP
- Def: Um operador é **associativo à esquerda** se os operandos são agrupados da esq para dir, e é **associativo à direita** se os operandos são agrupados da dir para esq.

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

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$

+ e*: associativos à esquerda

- Está relacionada com a posição da recursão nas regras que permitem um operador ser aplicado mais de uma vez.

Pascal

- 1) Expressões parentizadas são resolvidas primeiro
- 2) Not associativos a direita
- 3) * / div mod and associativos a esquerda
- 4) + - or " "
- 5) < > <> >= <= = ocorrem uma vez

$EXP \rightarrow ES \langle \rangle ES \mid ES$ (uma vez)

$ES \rightarrow ES + T \mid T$ (esq)

$T \rightarrow T * F \mid F$ (esq)

$F \rightarrow \text{not } F \mid U$ (dir)

$U \rightarrow \langle \text{inteiro} \rangle \mid \langle \text{real} \rangle \mid (EXP)$

- Os níveis de **prioridade** indicam a quais operadores é permitido agrupar seus operandos primeiro (resolver primeiro).

Not

* / div mod and

+ - or

< > <> >= <= =



Prioridade

Precedência

Linguagens Inerentemente ambíguas

É simples encontrar um exemplo de *GLC* ambígua. Na gramática abaixo para a sentença "a" temos 2 árvores

$$S \rightarrow A \mid B$$
$$A \rightarrow a$$
$$B \rightarrow a$$

Mas não é tão simples exibir uma *LLC* inerentemente ambígua

Def 😞: Uma *LLC* é inerentemente ambígua se toda gramática que a gera é ambígua

$$L = \{a^i b^j c^k \mid i, j, k \geq 1 \text{ e } i = j \text{ OU } j = k\}$$

$S \rightarrow abc \mid aRbI \mid YbWc$

$I \rightarrow Ic \mid c$

$R \rightarrow ab \mid aRb$

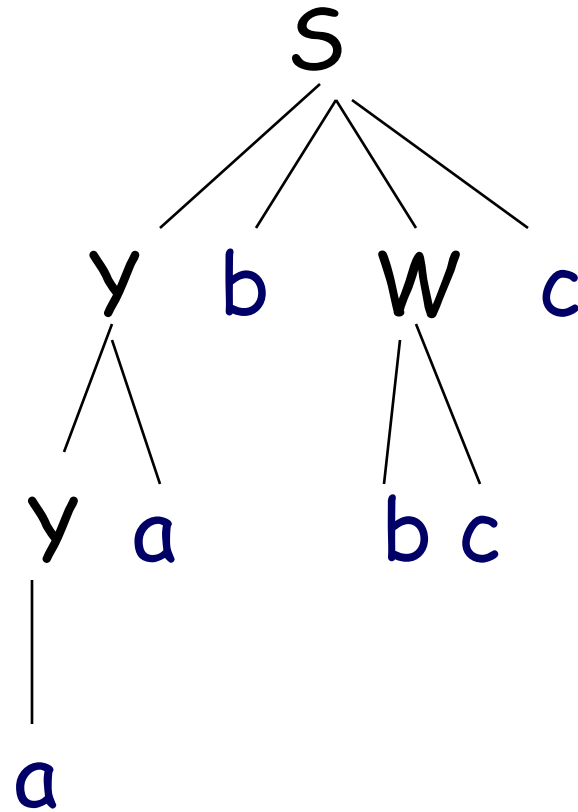
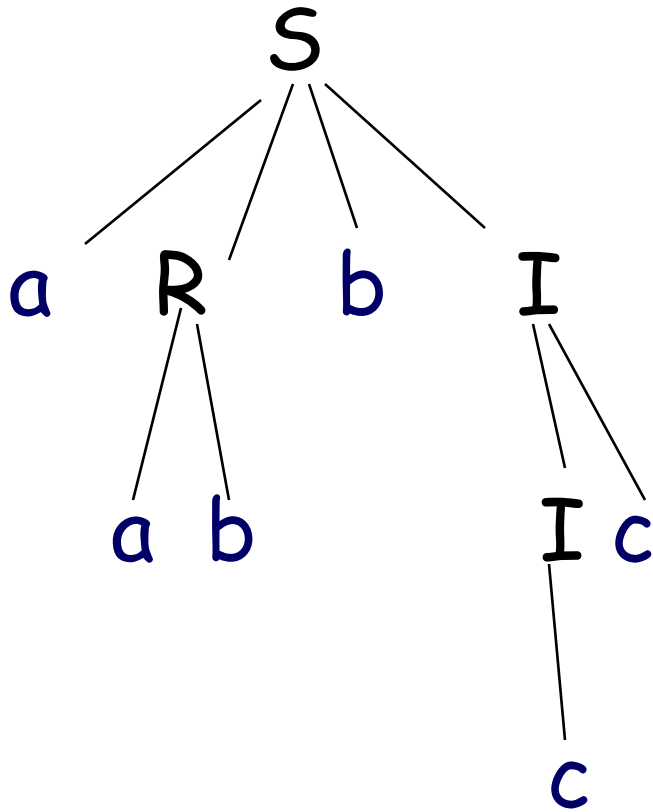
$Y \rightarrow Ya \mid a$

$W \rightarrow bc \mid bWc$

O fato de ser inerentemente ambígua decorre de que toda gramática que gera L gera cadeias para $i=j$ por um processo diferente do qual usa para gerar as cadeias para $j=k$.

É impossível não gerar algumas cadeias para as quais $i=j=k$ por ambos os processos.

- Exemplo: aabbcc



Propriedades de Gramáticas Ambíguas

Teo ☹: Não existe um algoritmo tal que, dada uma *GLC* qualquer, retorne a resposta sim, se ela for ambígua, ou não, se ela não for ambígua.

MAS

- ☺ em casos particulares, nós podemos reconhecer a ambiguidade e remove-la "à mão"
- ☺ E podemos utilizar classes mais restritas de *GLC* que por definição não são ambíguas (ex. $LL(K)$)

Este problema (decidir se uma *GLC* é ambígua) é **parcialmente decidível**, pois, para determinadas gramáticas, o procedimento pára e diz sim, MAS pode não parar para outros casos.

Exemplos de produções ambíguas

1. $A \rightarrow AA$

2. $A \rightarrow A \alpha A$

3. $A \rightarrow \alpha A \mid A\beta$

4. $A \rightarrow \alpha A \mid \alpha A\beta A$

Quais Gramáticas são ambíguas?

(1) $S \rightarrow bA \mid aB$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

(2) A gramática usada para gerar expressões aritméticas na notação **posfix** na linguagem APL:

$S \rightarrow SS + \mid SS - \mid SS * \mid x \mid y$

ou na notação **prefix**:

$S \rightarrow +SS \mid -SS \mid *SS \mid x \mid y$

(3) A gramática para gerar parênteses aninhados:

$S \rightarrow (S) \mid () \mid SS$

(4) A gramática que define os operadores lógicos and e or

$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid (E) \mid a$

Exercício para casa

- Identifiquem a parte de definição de Expressões nas linguagens de programação escolhidas e levantem as regras de precedência, prioridade e associatividade dos operadores.

Aplicações de GLCs

- **Analísadores Sintáticos**
 - *YACC* (Unix): tomam uma *GLC* como entrada e produzem um analisador sintático da linguagem correspondente.
- **DTD/XML**
 - notação para descrever a estrutura de documentos: aninhamento de tags semânticas. Uma *DTD* é uma *GLC* cuja linguagem é uma classe de documentos inter-relacionados.