Gramáticas e Linguagens

Livres de Contexto

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 $w= 0^n10^n$. 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 -> ε
2. P -> 0
3. P -> 1
4. P -> 0P0
5. P -> 1P1

Base

Indução

Uma Gramática Livre de Contexto para Palíndromos

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

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

$$cabeça\ (Vn) \rightarrow corpo\ (Vn \cup Vt)^*$$

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

Assumimos Vn $\cap$ Vt = $\varnothing$. Convencionamos que **Vn U Vt = V**
Cada produção P tem a forma:

$$A \rightarrow \beta \qquad A \in Vn; \quad \beta \in V^*$$

S ϵ Vn denota a principal categoria gramatica de G; é dito o símbolo inicial ou o axioma da gramática.

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

P: {S -> AB
A -> a
B -> b}

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 -> I | E + E |E * E|(E)
2. I -> a|b|Ia|Ib|I0|I1

➔I = Linguagem regular:
(a+b)(a+b+0+1)*

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.: $L(G_{pal})$, 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 $L(G_{pal})$ 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 $L(G_{pal})$.

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=>* w em qualquer desses casos básicos.

Indução: Suponha verdade para $|w|=n\geq 2$. Ou seja, $P \Rightarrow^* w$.

Seja $|w| = n+2$. 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 $L(G_{pal})$.

(Somente se) Supomos que $w \in L(G_{pal})$ 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 a hipótese seja verdadeira para todas as derivações de n passos, ou seja, se P=>*x em n passos, então x é palíndromo.

Vamos analisar o caso de uma derivação de (n+1) passos, $n \geq 1$:

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

P -> 0P0 =>* 0x0 = w ou

P-> 1P1 =>* 1x1 = w

Em qualquer dos casos, P=>* x em n passos.

Pela hipótese indutiva, sabemos que x é um palíndromo. Então 0x0 e 1x1 também são palíndromos, o que completa a prova.

Exemplo de GLC

G1 = ({S}, {0,1}, P1, S)
P1: { 1. S -> 0S1
2. S -> 01 }

Qual é a linguagem gerada por G1? Aplicamos o processo de derivação para descobrir L(G1).

G1

- *A menor cadeia gerada é* 01: S -> 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, $L(G1) = \{0^n1^n \mid n >= 1\}$

  ou $S \Rightarrow^* 0^n1^n (n >= 1)$

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 = (Vn,Vt,P,S) é uma árvore rotulada ordenada em que cada nó é rotulado por um símbolo de $Vn \cup Vt \cup \varepsilon$.
- Se um nó interno é rotulado com A e seus descendentes diretos são rotulados com $X_1, X_2, ..., X_n$ então

  $A \rightarrow X_1X_2...X_n$ é uma produção de P.

Uma árvore rotulada ordenada D é uma árvore de derivação para uma GLC G(A) = (Vn,Vt,P,A) se

(1) A raiz de D é rotulada com A

(2) Se D1, ...Dk são as subárvores de descendentes diretos da raiz e a raiz de Di é rotulada com Xi,
então A => X1,...Xk é uma produção em P.
Di deve ser a árvore de derivação para G(Xi) = (Vn,Vt,P,Xi) se Xi é não-terminal, e Di é um nó simples rotulado com Xi se Xi é terminal.

(3) Se D1 é a única subárvore da raiz D e a raiz de D1 é rotulada com λ então A -> ε 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.

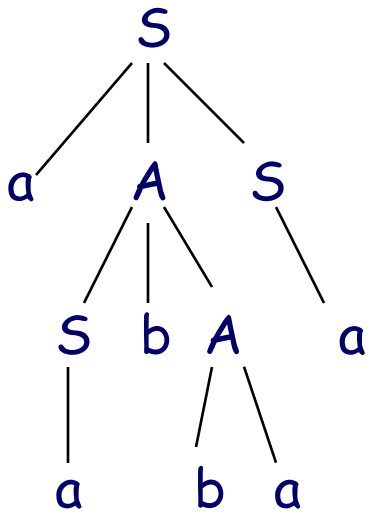
Uma sentença está representada na árvore de derivação fazendo-se a leitura das folhas (nós sem descendentes) da esquerda para direita.

Exemplos

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

P: S-> aAS | a

A -> SbA | ba | SS



- Seja Ga = {{E,T,F}, {+,*,(,),a},P,E)

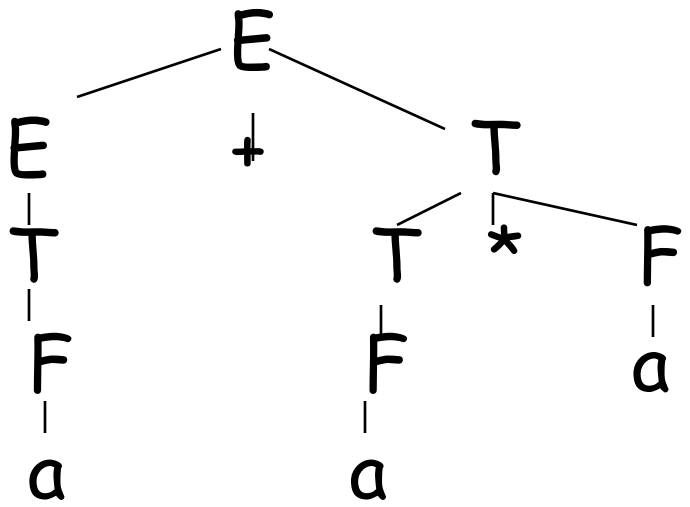
P: E -> E + T | T

T -> T * F | F

F -> (E) | 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:

E=>E+T=>T+T=>F+T=>a+T=>a+T*F=>a+F*F
=>a+a*F=>a+a*a

- Derivação mais à direita:

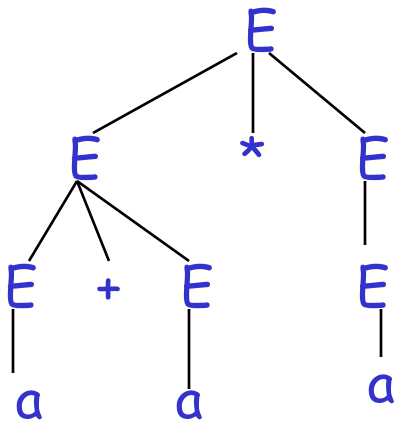
E=>E+T=>E+T*F=>E+T*a=>E+F*a=>E+a*a
=>T+a*a=>F+a*a=>a+a*a

Agora considere a gramática Gb:

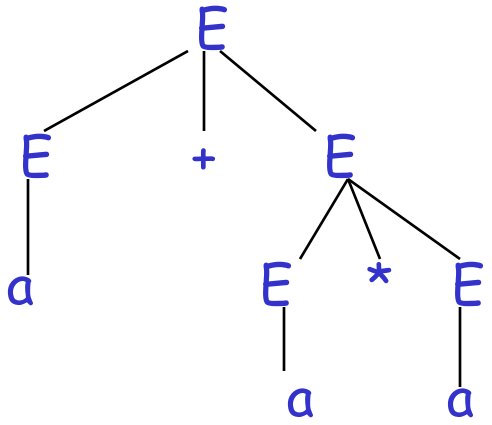
1. E -> E + E |E * E|(E)|a

Não é difícil ver que L(Ga)=L(Gb).

Faça a árvore de derivação de a+a*a para Gb:



Não é única!!



$E \underset{lm}{=>} E*E => E+E*E => a+E*E => a+a*E => a+a*a$

$E \underset{lm}{=>} E+E => a+E => a+E*E => a+a*E => a+a*a$

➔Gramática Ambígua

De fato:

- Ga indica que o operador * tem precedência sobre o operador +
- Por Gb, 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 implicam interpretações distintas.

Ambiguidade e Derivações mais à Esquerda

Teorema: Para cada gramática $G=(Vn, Vt, P, S)$ e cadeias w em Vt^*, 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.

Ambiguidade nas GLC

- Um requisito importante de uma LP é que ela não seja ambígua.

- O mais famoso caso de ambiguidade é o else pendente, presente na especificação de muitas LP.

- Seja a gramática:

C -> if b then C else C
C -> if b then C
C -> s

Ela é ambígua desde que a cadeia

If b then if b then s else s

Pode ser interpretada como

(i) If b then (if b then s else s)

Ou

(ii) If b then (if b then s) 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 -> if b then C1 | if b then C2 else C1 | s

C2 -> if b then C2 else C2 | 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 then-else. Assim, a interpretação (ii) nunca ocorre.

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

Obs.: se essa for a interpretação desejada, então, nas LPs, usamos begin-end.

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) caracteriza 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 -> E + T | T

  T -> T * F | F

  F -> (E) | 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 -> ES <> ES | ES (uma vez)
ES -> ES + T | T (esq)
T -> T * F | F (esq)
F -> not F | U (dir)
U -> <inteiro> | <real> | (EXP)

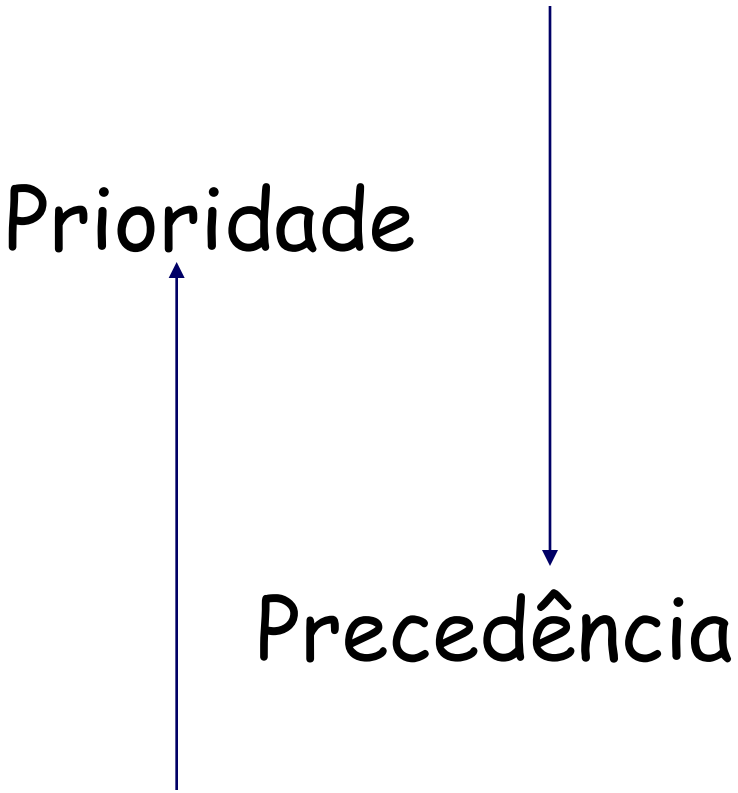
- Os níveis de prioridade indicam a quais operadores é permitido agrupar seus operandos primeiro (resolver primeiro).

Not

* / div mod and

+ - or

< > <> >= <= =



Linguagens Inerentemente ambíguas

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

S -> A | B

A -> a

B -> 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

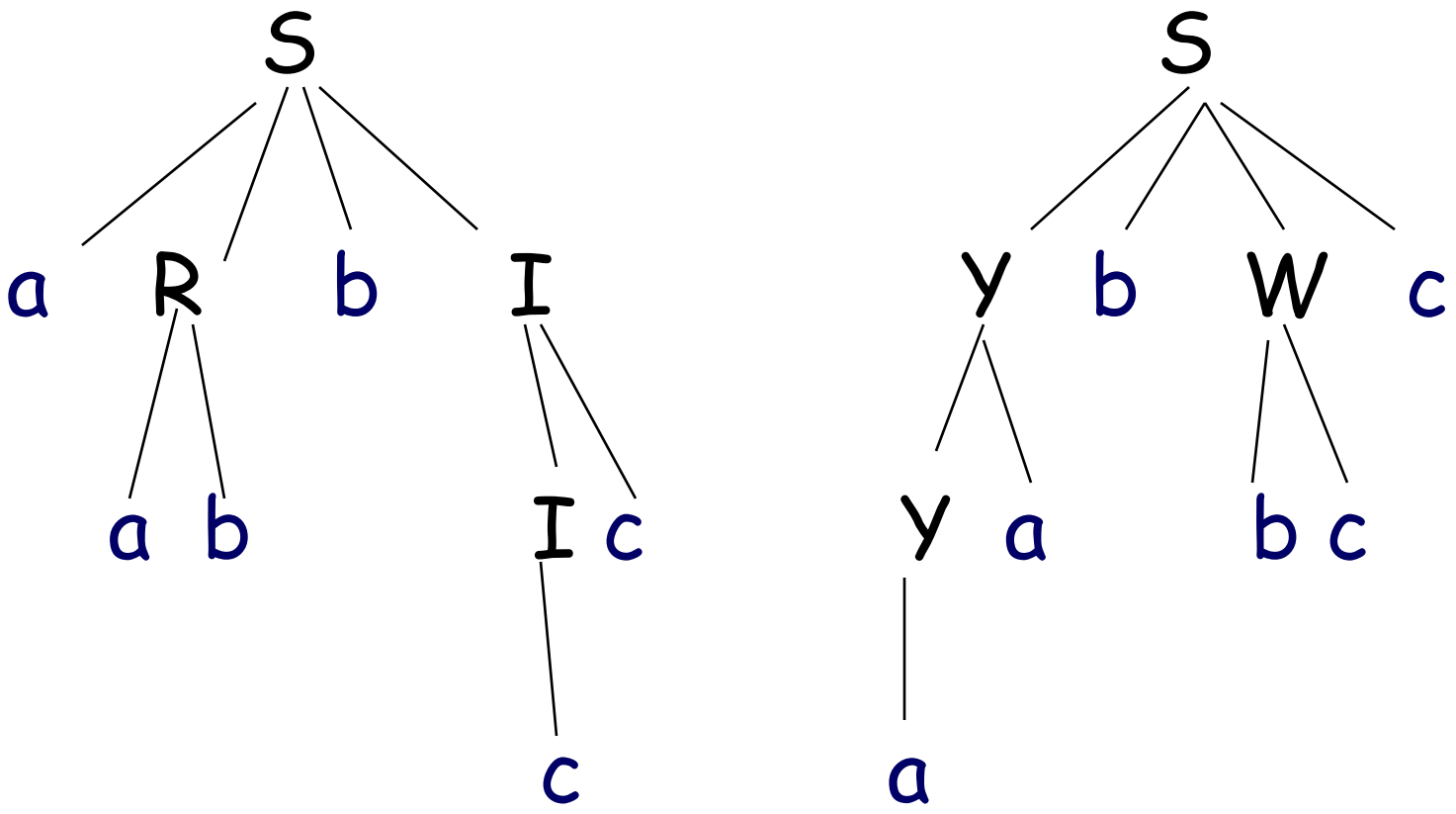
$L = \{a^i b^j c^k \mid i,j,k >= 1 \text{ e } i = j \text{ OU } j = k\}$

S -> abc | aRbI | YbWc
I -> Ic | c
R -> ab | aRb
Y -> Ya | a
W -> bc | 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 -> bA | aB
A -> a | aS | bAA
B -> b | bS | aBB

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

S -> SS + | SS - | SS * | x | y

ou na notação prefix:

S-> +SS | -SS | *SS | x | y

(3) A gramática para gerar parênteses aninhados:

S -> (S) | ( ) | SS

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

E -> E or E | E and E | (E) | 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

- Analisadores 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.