Linguagens e Gramáticas

Linguagens Formais
Hierarquia de Chomsky

Já vimos que

Linguagem é um conjunto de *cadeias* de símbolos sobre um alfabeto/vocabulário, V. É um subconjunto específico de V*. Estas cadeias são denominadas *sentenças* da linguagem, e são formadas pela justaposição de elementos individuais, os símbolos da linguagem.

Pode-se representar uma linguagem:

(1) como um conjunto finito ou infinito de cadeias

Ex: Linguagem dos números pares; $L=\{ 0^n 1^n \mid n \geq 1\}$

(2) por meio de uma Máquina de Turing que a reconheça

(3) Por meio de uma Gramática que a gere

A Gramática é o formalismo gerativo de linguagens, enquanto que os automatos (a MT é um automato) são reconhecedores de linguagens.

Exemplo 1:

Vn

Vt

Axioma

G1 = ({S,A,B}, {a,b}, P,S)

P = { 1. S -> aB
2. S -> bA
3. A -> a
4. A -> aS
5. A -> bAA
6. B -> b
7. B -> bS
8. B -> aBB }

P: Regras de Produção

Como G gera uma linguagem? Por um processo de substituição ou derivação de símbolos. As cadeias ∈ Vt* geradas formam a linguagem L(G).

$S \to^1 aB \to^8 aaBB \to^7 aabB \to^7 aabb$
$S \to^2 bA \to^4 baS \to^1 baaB \to^7 baab$
etc.

Definição:

Formalmente, as gramáticas são caracterizadas como quádruplas ordenadas

G = (Vn, Vt, P, S)

onde:

1. Vn representa o vocabulário não terminal da gramática. Este vocabulário corresponde ao conjunto de todos os símbolos dos quais a gramática se vale para definir as leis de formação das sentenças da linguagem.

2. Vt é o vocabulário terminal, contendo os símbolos que constituem as sentenças da linguagem. Dá-se o nome de terminais aos elementos de Vt.

3. P representa o conjunto de todas as leis de formação utilizadas pela gramática para definir a linguagem.

Para tanto, cada construção parcial, representada por um não-terminal, é definida como um conjunto de regras de formação relativas à definicão do não-terminal a ela referente. A cada uma destas regras de formação que compõem o conjunto P dá-se o nome de produção da gramática.

Assumimos Vn ∩ Vt = ∅. Convencionamos que Vn U Vt = V
Cada produção P tem a forma:

$\alpha \rightarrow \beta$ $\alpha \in V^+$;(qualquer cadeia não nula de V) e
$\beta \in V^*$ (qualquer cadeia de V, incluindo a nula)

4. S ϵ Vn denota a principal categoria gramatica de G; é dito o símbolo inicial ou o axioma da gramática. Indica onde se inicia o processo de geração de sentenças.

Notação/Convenções

- Letras do alfabeto latino maiúsculas {A,B,..Z}: variáveis
- Letras do começo do alfabeto latino minúsculas {a,b,c,...}: terminais
- Letras do fim do alfabeto latino minúsculas {t,u,v,x,z}: cadeias de terminais
- Letras gregas minúsculas {α,β,γ,**δ,ε,** ..., ω}: cadeias de terminais e não terminais.
- Regras de produção com mesmo lado esquerdo são simplificadas com a notação | **(ou)**

G1 = ({S,A,B}, {a,b}, P,S)

P1 = {S -> aB | bA

A -> a | aS | bAA

B -> b | bS | aBB }

Definida uma gramática G, qual é a linguagem gerada por ela?

Sejam as relações

=> (deriva/gera diretamente) e

$=>^*$ (deriva/gera)

definidas entre as cadeias de V*

Def.1. Se $\alpha \rightarrow \beta$ é uma produção de P e γ e δ são cadeias quaisquer de V*, então

$$\gamma\ \alpha\ \delta \Rightarrow \gamma\ \beta\ \delta$$

Ou: α deriva/gera β (β é derivado de α) por uma única produção, não importa o contexto em que aparecem.

No Ex.1.: S => aB ; aB => aaBB; B=>b ou

S => aB =>aaBB=> aabB=> aabb

Def.2. Suponha que $\alpha_1\ \alpha_2\ \alpha_3\ \ldots\ \alpha_m$ são cadeias de V* e

$$\alpha_1 \Rightarrow_G \alpha_2,\quad \alpha_2 \Rightarrow_G \alpha_3,\quad \ldots,\quad \alpha_{m-1} \Rightarrow_G \alpha_m$$

Então dizemos que

$$\alpha_1 \Rightarrow^* \alpha_m$$

Ou: α_1 deriva/gera α_m aplicando-se um número qualquer de produções de P.

Por convenção $\alpha \Rightarrow^* \alpha$ para toda cadeia α.

No Ex.1.: S =>* ab;
S =>* aaBB;
baS =>* baab;
aB =>* abbA

Def.3. Toda cadeia derivada/gerada do símbolo inicial S é chamada uma **forma sentencial**.

Ou seja, uma cadeia $\alpha \in V^*$ é uma forma sentencial se $S \Rightarrow^* \alpha$

No Ex.1: aB, AB, S, ab são formas sentenciais.

Def.4. Uma forma sentencial, α, é uma **sentença** de G se for composta apenas de símbolos terminais.

Ou: $\alpha \in V^*$ é uma sentença se
(a) $S \Rightarrow^* \alpha$ (for forma sentencial) e
(b) $\alpha \in Vt^*$ (só tem terminais).

Assim, as sentenças são as cadeias de terminais geradas pela gramática (por seu símbolo inicial).

Def.5. A Linguagem L gerada por uma gramática G é definida como o conjunto de sentenças de G. Ou seja,

$L(G) = \{x \mid x \in Vt^* \text{ e } S \Rightarrow^* x\}$ ou {x | x é sentença de G}

1. A cadeia consiste somente de terminais
2. A cadeia é derivada a partir do símbolo inicial da gramática

Def.6. Duas gramáticas G1 e G2 são equivalentes sse

$L(G1) = L(G2)$

Exemplos de Gramáticas

G2 = ({S}, {0,1}, P1, S)
P: { 1. S -> 0S1
2. S -> 01 }

Qual é a linguagem gerada por G1? Aplicamos o processo de derivação para saber L(G1), que é o processo de obtenção de cadeias a partir de uma gramática.

G3 = ({S,B,C}, {a,b,c}, P2, S)
P: { 1. S -> aSBC 2. S -> aBC
3. CB -> BC 4. aB -> ab
5. bB -> bb 6. bC -> bc
7. cC -> cc}
L(G3) = ?

G2

- A menor cadeia gerada é 01: $S \Rightarrow^2 01$

- 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, $L(G2) = \{0^n1^n \mid n >= 1\}$

  ou $S \Rightarrow^* 0^n1^n$

G3

- A menor cadeia gerada é abc: $S =>^2 aBC =>^4 abC =>^6 abc$
- Usamos 1 n-1 vezes: $S =>^* a^{n-1}S(BC)^{n-1}$
- Usamos a 2 uma vez: $S =>^* a^n(BC)^n$
- A 3 permite trocar B com C para que B´s precedam os C´s
  - Para n = 2 aaBCBC => aaBBCC (usamos a regras 3 1 vez)
  - Para n = 3 aaaBCBCBC => aaaBBCCBC => aaaBBCBCC => aaaBBBCCC (usamos a regra 3 3 vezes)
  - Para n = 4 aaaaBCBCBCBC => aaaBBCCBCBC => aaaaBBCBCCBC => aaaaBBBCCCBC =>aaaaBBBCCBCC => aaaaBBBCBCCC => aaaaBBBBCCCC (usamos a regra 3 5 vezes);
  - Para n = 5 usamos a 3 10 vezes.
- Assim $S =>^* a^nB^nC^n$
- Usamos a 4 uma vez: $S =>^* a^nbB^{n-1}C^n$
- Aplicamos a 5 n-1 vezes: $S =>^* a^nb^nC^n$
- Aplicamos a 6 uma vez: $S =>^* a^nb^ncC^{n-1}$
- Aplicamos a 7 n-1 vezes: $S =>^* a^nb^nc^n$

$$L(G3) = \{a^nb^nc^n \mid n >= 1\}$$

Tipos de Gramáticas

- Chamamos o tipo de gramática que definimos **de tipo 0 ou com Estrutura de Frase ou Irrestritas.**

$\alpha \rightarrow \beta \qquad \alpha \in V+; \quad \beta \in V^*$

- Não há restrições nas regras de produção.

G4= ({S,A,B,C,D,E}, {a}, P,S)

P = { 1. S-> ACaB
2. Ca -> aaC
3. CB -> DB
4. CB -> E
5. aD -> Da
6. AD -> AC
7. aE -> Ea
8. AE -> λ }

L(G4)=?

- **Menor cadeia: aa**

S =>[1] ACaB =>[2] AaaCB =>[4] AaaE =>[7] AaEa =>[7] AEaa =>[8] aa

- **A e B servem como marcadores da esq e dir para as formas sentenciais.**

- **C é o marcador que se move através da cadeia de a´s entre A e B, dobrando seu número pela produção 2.**

- **Quando C alcança o marcador à direita B, ele se torna um D ou E pela produção 3 ou 4.**

- **Se um D é escolhido, então ele migra à esquerda pela produção 5 até que o marcador à esq, A, seja alcançado.**

- **Nesse ponto, D se torna C de novo pela produção 6 e o processo recomeça.**

- **Se um E é escolhido, o marcador à direita (B) é consumido.**

- **O E migra à esquerda pela produção 7 e consome o marcador à esq pela produção 8.**

$$L(G4) = \{a^{2n} \mid n \text{ é um inteiro positivo}\}$$

Classes Gramaticais

Conforme as restrições impostas ao formato das produções de uma gramática, a classe de linguagens que tal gramática gera varia correspondentemente. A teoria mostra que há quatro classes de gramáticas capazes de gerar quatro classes correspondentes de linguagens, de acordo com a denominada Hierarquia de Chomsky:

Gramáticas Recursivamente Enumeráveis ou Irrestritas ou Tipo 0

Gramáticas Sensíveis ao Contexto ou Tipo 1

Gramáticas Livres de Contexto ou Tipo 2

Gramáticas Regulares ou Tipo 3

Linguagens LRE

As linguagens geradas pelas Gramáticas com Estrutura de Frase ou do Tipo 0 são chamadas de Linguagens Recursivamente Enumeráveis (LRE) ou Linguagens do Tipo 0.

São as linguagens para as quais há uma Máquina de Turing que as reconhece.

Gramáticas Sensíveis ao/Dependentes de Contexto ou Tipo 1

Se às regras de substituição for imposta a restrição de que nenhuma substituição possa reduzir o comprimento da forma sentencial à qual a substituição é aplicada, cria-se uma classe de gramáticas ditas sensíveis ao contexto.

As gramáticas que obedecem a estas restrições pertencem, na hierarquia de Chomsky, ao conjunto das Gramáticas Sensíveis ao Contexto (GSC) ou do Tipo 1.

Para as GSC, as produções são todas da forma

$\alpha \rightarrow \beta$, com $|\alpha| <= |\beta|$

onde $\alpha, \beta \in (Vn \cup Vt)^+$

(produções não decrescentes)

Alguns autores colocam as produções de uma GSC como:

$\alpha 1 A \alpha 2 \rightarrow \alpha 1 \beta \alpha 2$ com $\alpha 1, \alpha 2, \beta \in V^*$, $\beta <> \lambda$ e $A \in Vn$

para motivar o nome "sensível ao contexto" desde que a produção $\alpha 1 A \alpha 2 \rightarrow \alpha 1 \beta \alpha 2$ **permite que A seja trocado por β no contexto de α1 e α2.**

G4 é GSC; G3 e suas equivalentes abaixo também são GSC:

G5 = ({A,B,C}, {a,b,c},P,A)

P = { A -> abc
A -> aBbc
Bb -> bB
Bc -> Cbcc
bC -> Cb
aC -> aaB
aC -> aa }

G6 = ({S,C}, {a,b,c},P,S)

P = { S -> abc
ab -> aabbC
Cb -> bC
Cc -> cc }

Linguagens Sensíveis ao Contexto - LSC

As linguagens geradas pelas Gramáticas Sensíveis ao Contexto ou do Tipo 1 são chamadas de Linguagens Sensíveis ao Contexto (LSC) ou Linguagens do Tipo 1.

Resultado 1:

Toda gramática do tipo 1 é também do tipo 0.

Corolário 1:

Toda LSC é também uma LEF (mas nem toda LEF é LSC).

Gramáticas Livres de Contexto ou Tipo 2

As Gramáticas Livres de Contexto (GLC) ou do Tipo 2 são aquelas cujas regras de produção são da forma:

$A \rightarrow \alpha$ onde $A \in Vn$, $\alpha \in V+$

Ou seja, quando do lado esquerdo da regra há apenas um símbolo não-terminal

A gramática G1, do Ex 1, é uma GLC.

G1 = ({S,A,B}, {a,b}, P,S)

P = {S -> aB | bA

A -> a | aS | bAA

B -> b | bS | aBB }

L(G1) = ?

- Menores cadeias: ab e ba

S => aB => ab

S => b*A* => ba

S => aB => abS => abb*A* => abba

=> abaB => abab

=> aaBB => aabb

S => b*A* => baS => baaB => baab

=> bab*A* => baba

=> bb*AA* => bbaa

L(*G1*) = {w $\in$ {a,b}$^+$ | nro(a) = nro(b)}

Todas as combinações de cadeias em V^+ com nro(a) = nro(b)

Linguagens Livres de Contexto (LLC)

As linguagens geradas pelas Gramáticas Livres de Contexto ou do Tipo 2 são chamadas de Linguagens Livres de Contexto (LLC) ou Linguagens do Tipo 2.

Resultado 2:

Toda gramática do tipo 2 é também do tipo 1.

Corolário 2:

Toda LLC é também uma LSC (mas nem toda LSC é uma LLC).

BNF

Outra maneira de se representar as Gramáticas Livres de Contexto é através da Forma Normal de Backus.

Neste caso, -> é substituído por ::= e os não terminais são ladeados por < >

No caso de repetições de lado esquerdo:

<A> ::= a1
<A >::= a2
:
<A> ::= an
escreve-se: <A> ::= a1| a2| ...| an

Os símbolos <,> , ::=, | formam a metalinguagem, ou seja, são símbolos que não fazem parte da linguagem mas ajudam a descrevê-la.

Exemplo:

G7 = {Vn, Vt, P, S} onde:

Vn = {<sentença, <sn>, <sv>, <artigo>, <substantivo>, <verbo>}

Vt = {o, a, peixe, comeu, isca}

S = <sentença>

P = {

1. <sentença> ::= <sn> <sv>
2. <sn> ::= <artigo> <substantivo>
3. <sv> ::= <verbo> <sn>
4. <artigo> ::= o|a
5. <verbo> :: = mordeu
6. <substantivo> ::= peixe|isca }

Exercícios:

a) verifique se a cadeia "a isca mordeu o peixe" é uma sentença de L(G7).

b) Dê exemplos de sentenças de L(G7).

Mais GLC:

G8= ({S}, {a, +, *, (,)}, P, S)
P = {
S -> S * S
S -> S + S
S -> (S)
S -> a }

L(G8)= conjunto das expressões aritméticas envolvendo *, +, ( ) e a.

Um exemplo de cadeia formada por esta gramática é a * (a + a).

Processo inverso: Dada uma L(G) definir a gramática G.

$L(G9) = \{a^m b^n \mid m \geq 1,\ n \geq 1\}$

L(G9) = $\{a^m b^n \mid m \geq 1,\ n \geq 1\}$ ou a^+b^+

Resp.:
G9=({S, A, B}, {a, b}, P, S)
P = {S -> AB
A -> aA | a
B -> bB | b }

Obs.: Caso geral:

Se $A \rightarrow \alpha A \mid \beta$ então $A \Rightarrow \alpha^* \beta$

$L(G10) = \{a^n b^n \mid n \geq 1\}$

$L(G10) = \{a^n b^n \mid n \geq 1\}$

$G = (\{S\}, \{a,b\}, P, S)$

$P = \{S \rightarrow aSb \mid ab\}$

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.

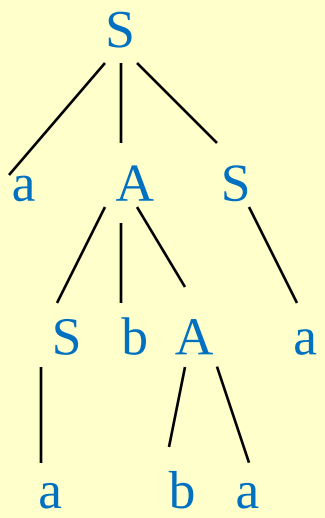
Uma sentença está representada na árvore de derivação fazendo-se a leitura das folhas (nós sem descendentes) da esquerda para direita

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) Seja a sentença a + a *a. Mostre a derivação mais à esquerda e a mais à direita.

- Derivação mais à esquerda:

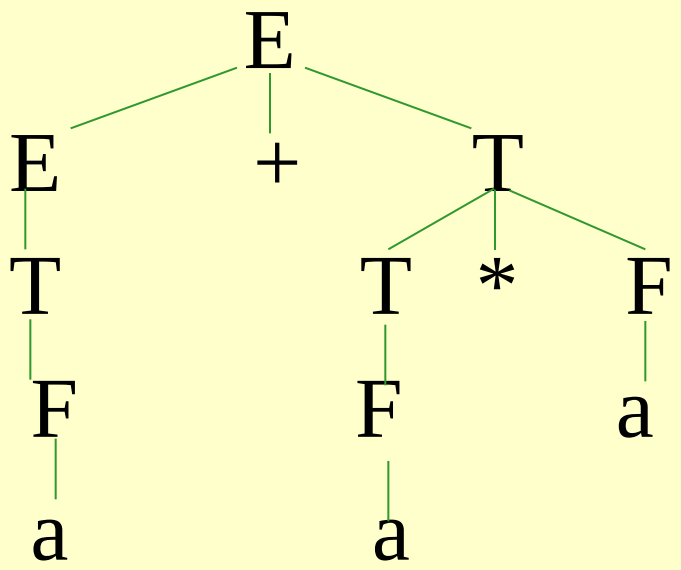
E=>E+T=>T+T=>F+T=>a+T=>a+T*F=>a+F*F=>a+a*F=>

- 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=>

2) Agora, mostre a árvore de derivação para

a + a * a

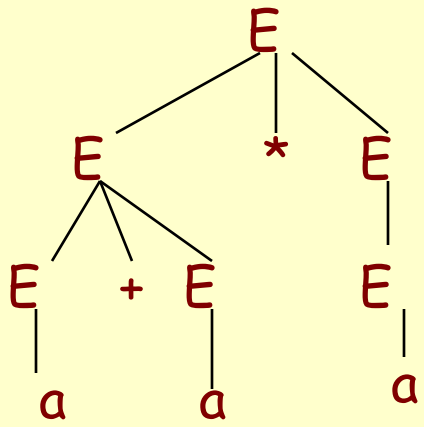


Única Árvore de Derivação

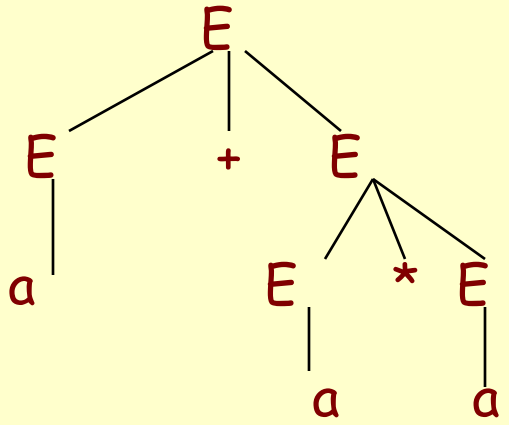
Agora considere a gramática Gb:

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=>E*E=>E+E*E=>a+E*E=>a+a*E=>a+a*a

Der. esq.

E=>E+E=>a+E=>a+E*E=>a+a*E=>a+a*a

Der. esq.

➔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

S : outro comando qualquer

(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

S : outro comando qualquer

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 de gramáticas de operadores:
  - 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

Ex.: a + a + a (1º.; 2º.)

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

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

not

* / div mod and

+ - or

< > <> >= <= =

Prioridade

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

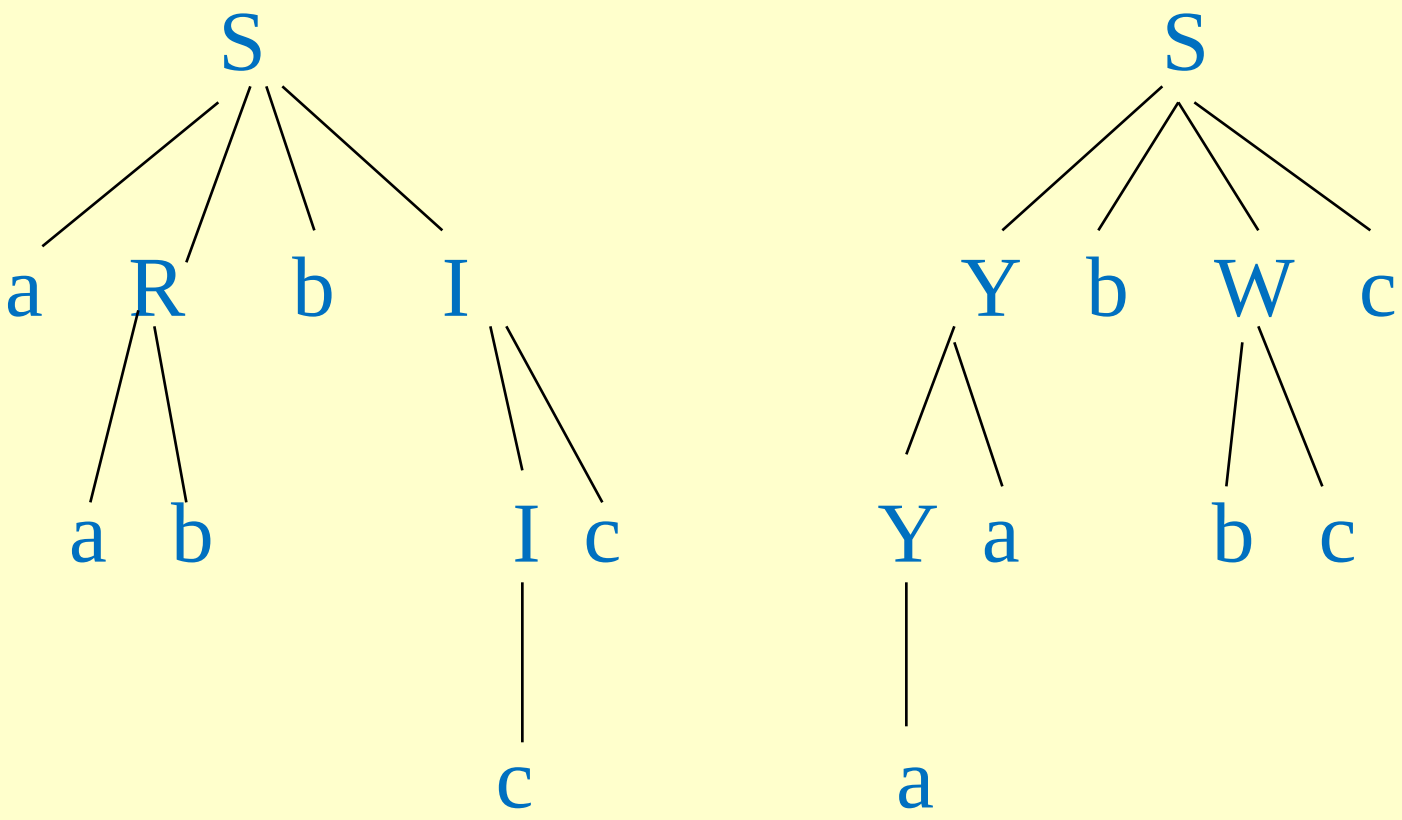
Y -> Ya | a

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 -> AA
2. A -> A α A
3. A -> αA | Aβ
4. A -> αA | αAβA

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 na linguagem APL:

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

ou na notação :

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

Gramáticas Regulares ou Tipo 3

Aplicando-se mais uma restrição sobre a forma das produções, pode-se criar uma nova classe de gramáticas, as Gramáticas Regulares (GR). Nas GRs, as produções são restritas às formas seguintes:

A -> aB ou A -> a (linear à direita)

ou

A -> Ba ou A -> a (linear à esquerda)

onde A,B $\in$ Vn e a $\in$ Vt

Linguagens Regulares (LR)

As linguagens geradas pelas Gramáticas Regulares ou do Tipo 3 são chamadas de Linguagens Regulares (LR) ou Linguagens do Tipo 3. São as linguagens mais simples.

Resultado 3:

Toda gramática do tipo 3 é também do tipo 2.

Corolário 3:

Toda LR é também uma LLC (mas nem toda LLC é LR).

Exemplo 1:

G11 = ({S}, {a, b}, P, S)
P = {
S -> aS
S -> b }

Exemplo 1:

G11 = ({S}, {a, b}, P, S)
P = {
S -> aS
S -> b }

Resp.: L(G11) = {$a^n b$| n ≥0} ou a*b

Exemplo 2:

G12 = ({S, A}, {a, b, c}, P, S)
P = {
S -> aS | bA
A -> c }

Exemplo 2:

G12 = ({S, A}, {a, b, c}, P, S)
P = {
S –> aS | bA
A –> c }

Resp.: L(G12) = {$a^n bc$ | n ≥ 0} ou a*bc

Formato BNF – Backus-Naur Form

Exemplo 3:

G13 = ({<Dig>, <Int>}, {+, -, 0, ..., 9}, P, <Int>)
P = {
<Int> ::= +<Dig> | -<Dig>
<Dig> ::= 0<Dig>|1<Dig>|...|9<Dig>|0|1|2|...|9
}

Formato BNF – Backus-Naur Form

Exemplo 3:

G13 = ({<Dig>, <Int>}, {+, -, 0, ..., 9}, P, <Int>)
P = {<Int> ::= +<Dig> | -<Dig>
<Dig> ::= 0<Dig> | 1<Dig>|...| 9<Dig> | 0 | 1 |
2 |...|9 }

Resp.:
L(G13) = conj. números inteiros com sinal
$\pm(0+1+2\ldots+9)^+$ ou $\pm(0..9)^+$

Modifique para que o sinal do número seja opcional

Exemplo 3:

G13 = ({<Dig>, <Int>}, {+, -, 0, ..., 9}, P, <Int>)
P = {<Int> ::= +<Dig> | -<Dig> | <Dig>
<Dig> ::= 0<Dig> | 1<Dig>|...| 9<Dig> | 0 | 1 |
2 |...|9 }

Modifique para que a cadeia nula faça parte da linguagem

Exemplo 3:

G13 = ({<Dig>, <Int>}, {+, -, 0, ..., 9}, P, <Int>)
P = {<Int> ::= +<Dig> | -<Dig> | <Dig> | λ
<Dig> ::= 0<Dig> | 1<Dig>|...| 9<Dig> | 0 | 1 | 2 |...|9 }

Modifique para que a cadeia nula faça parte da linguagem

Exemplo 3: **OU**

G13 = ({<Dig>, <Int>}, {+, -, 0, ..., 9}, P, <Int>)
P = {<Int> ::= +<Dig> | -<Dig> | <Dig>
<Dig> ::= 0<Dig> | 1<Dig>|...| 9<Dig> | λ }

Exemplo 4:

G14 = ({A,B,C}, {0,1}, P, A)
P = { A -> 0B | 0
B -> 1C
C -> 0B | 0 }

Exemplo 4:

$G14 = (\{A,B,C\}, \{0,1\}, P, A)$

$P = \{ A \rightarrow 0B \mid 0$

$B \rightarrow 1C$

$C \rightarrow 0B \mid 0 \}$

Resp.:

$L(G14) = \{0(10)^*\}$

Linguagens Regulares e Expressões Regulares

- Expressões Regulares (ER) denotam conjuntos.
- Os conjuntos representados por ER coincidem com as Linguagens Regulares (LR).
- Assim, toda LR pode ser representada por uma ER.

Expressões Regulares (ER)

Uma ER sobre um alfabeto Σ é definida como:

a) $\varnothing$ é uma ER e denota a linguagem vazia
b) λ é uma ER e denota a linguagem contendo a palavra vazia, ie $\{\lambda\}$
c) Qualquer símbolo $x \in \Sigma$ é uma ER e denota a linguagem $\{x\}$
d) Se r e s são ER denotando as linguagens R e S então:
   - (r+s) ou (r|s) é ER e denota a linguagem $R \cup S$
   - (rs) é ER e denota a linguagem $RS = \{w=uv \mid u \in R \text{ e } v \in S\}$
   - (r^*) é ER e denota a linguagem R^*

Exemplos

- 00 é uma ER denotando a linguagem {00}
- (0+1)* denota a linguagem formada por todas as cadeias de 0´s e 1´s
- (0+1)* 00 (0+1)* denota todas as cadeias de 0´s e 1´s com ao menos dois 0´s consecutivos
- a+b*c denota um único a ou zero ou mais vezes b seguido de c

- (0+1)* 001 denota todas as cadeias de 0´s e 1´s terminadas em 001

- 0*1*2* denota qualquer número de 0´s seguido por qualquer número de 1´s seguido por qualquer número de 2´s

- 01* + 10* denota a linguagem consistindo de todas as cadeias que são um único 0 seguido por qualquer número de 1´s OU um único 1 seguido por qualquer número de 0´s.

Omissão de parênteses

- Para omitir parênteses devemos respeitar:
  - O fecho (*) tem prioridade sobre a concatenação (rs), que tem prioridade sobre a união.
  - A concatenação e a união são associadas da esquerda para a direita.
  - Ex: 01* + 1 é agrupado como (0(1*)) + 1 => L = {1, 0, 01, 011,...}

- Usamos parênteses quando queremos alterar a prioridade:

- (01)* + 1 => $L = \{1 \cup (01)^n \mid n >= 0\} = \{1, \lambda, 01, 0101, ...\}$

- 0(1* + 1) => $L = \{w \in \{0,1\}^* \mid$ w começa com 0 seguido de $1^n \mid n>=0\}$ ➔ Lei distributiva à esq = 01* + 01 = {0,01,011,0111,...}

Escreva a ER equivalente a:

- O conjunto de cadeias sobre {0,1} que termine com três 1´s consecutivos.

- O conjunto de cadeias sobre {0,1} que tenha ao menos um 1.

- O conjunto de cadeias sobre {0,1} que tenha no máximo um 1.

Escreva a ER equivalente a:

- O conjunto de cadeias sobre {0,1} que termine com três 1´s consecutivos.

$$(0+1)^*111$$

- O conjunto de cadeias sobre {0,1} que tenha ao menos um 1.

$$(0+1)^*1(0+1)^*$$

- O conjunto de cadeias sobre {0,1} que tenha no máximo um 1.

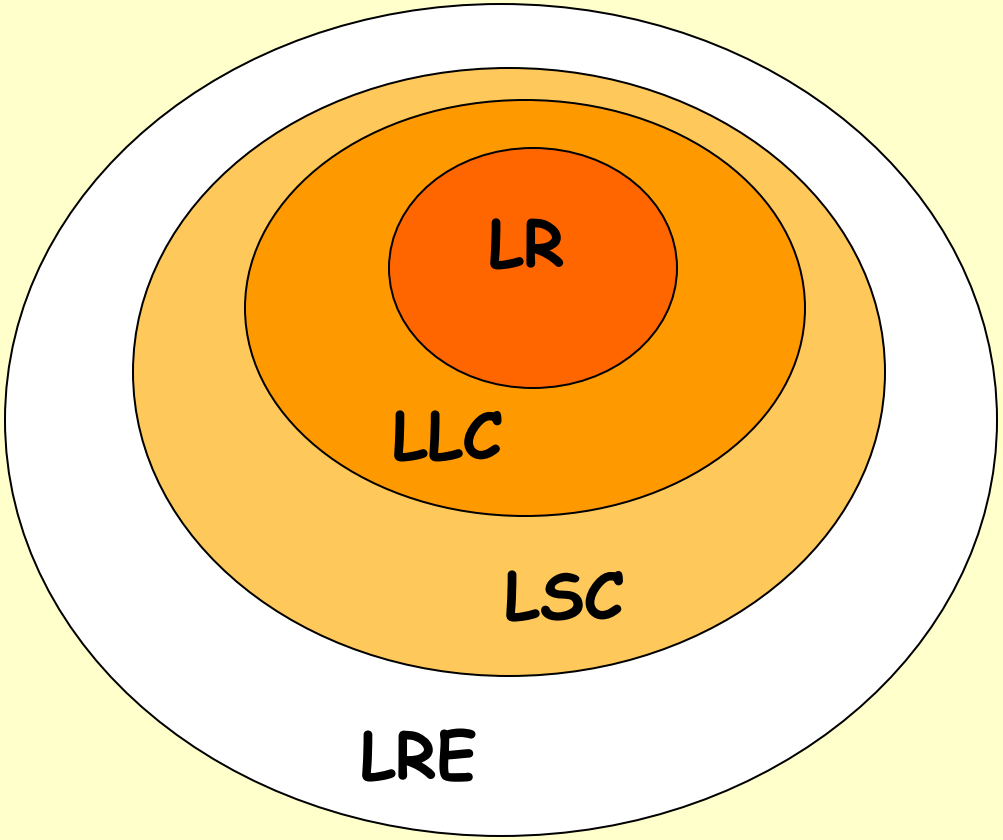
$$0^*(1+\lambda)0^*$$

Hierarquia de Chomsky

Em termos gerais, para $n \in \{0, 1, 2, 3\}$ pode-se afirmar que uma *linguagem* de qualquer tipo pode ser classificada também como sendo de tipo menor, de acordo com a Hierarquia de Chomsky.

Uma linguagem do tipo *n* é caracterizada pela existência de alguma gramática do tipo *n* que a descreva.

Hierarquia de Chomsky



LR = Linguagens Regulares
LLC = Linguagens Livres de Contexto
LSL =Linguagens Sensíveis ao Contexto
LEF = Linguagens Recursivamente Enumeráveis

Gramáticas e reconhecedores

Linguagens/Gramáticas	Reconhecedores
Rec. Enumerável	Máquina de Turing
Sensível ao contexto	Máquina de Turing com memória limitada
Livre de contexto	Autômato a pilha
Regular	Autômato finito

A partir de agora, vamos estudar as Linguagens (Problemas) mais simples: LR e LLC e seus reconhecedores.

Linguagens e Reconhecedores

Linguagem	Gramática	Reconhecedor	Tempo para reconhecer w; |w|=n
Tipo 0: Linguagens Computáveis ou Recursivamente Enumeráveis	Gramáticas com Estrutura de Frase	Máquinas de Turing	NP-completo
Tipo 1: Sensíveis ao Contexto	Gramáticas Sensíveis ao Contexto	Máquinas de Turing com memória limitada	Exponencial: $O(2^n)$
Tipo 2: Livres de Contexto	Gramáticas Livres de Contexto	Autômatos à Pilha	Polinomial Espaço: $O(n)$; Tempo: Geral: $O(n^3)$; Não-ambíguas: $O(n^2)$; Se P= A->aB ou A->Ba ou A->a: $O(n)$
Tipo 3: Conjuntos Regulares	Gramáticas Regulares	Autômatos Finitos	Linear: $O(n)$ e $O(\|E\|)$ (no tamanho do AF)

Classifique as gramáticas, dê a quádrupla e a L(G) e diga se são finitas/infinitas

1) E -> E + E | E - E | E * E | E / E | (E) | F
   F -> 0 | 1 | ... | 9

2) A -> BC
   BC -> CB
   B -> b
   C -> a

3) A -> 0A | B
   B -> 1B | λ

4) S -> 0A
   A -> 1S | 1

5) S -> 0A
   A -> 1B
   B -> 1S | 1

6) L(G) = {111(00)*}
G = ?
7) L(G) = {$a^n b^n c^*$ | n >= 1 e i >= 0}
G = ?
8) L(G) = {$a^* b^n c^n$ | n >= 1 e i>= 0}
G = ?
9) Utilize o software JFLAP com os exemplos acima