# Linguagens Regulares

Eduardo Furlan Miranda

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

# Linguagens Regulares (LR)



- Algumas abreviações
  - LC Livre de Contexto
  - LLC Linguagem Livre de Contexto
  - GLC Gramática Livre de Contexto
  - RLC Regra Livre de Contexto
  - SC Sensível ao Contexto
  - LSC Linguagem Sensível ao Contexto
  - GSC Gramática Sensível ao Contexto
  - RSC Regra Sensível ao Contexto
  - LR Linguagem Regular
  - GR Gramática Regular
  - MT Máquina de Turing
  - AF Autômato Finito

# Analisador Sintático (AS)

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



- Um reconhecedor (analisador sintático) para uma linguagem formal $L \subseteq \Sigma^*$ "está contido ou é igual a"
  - é um procedimento que ao ler qualquer palavra $\omega \in \Sigma^*$
    - indica se $\omega \in L$ ou se $\omega \notin L$
- Dado um símbolo $s$ e um alfabeto $\Sigma$ ,
  - saber se $s \in \Sigma$ e se $s \neq s_2 \in \Sigma$ deve ser um procedimento automático
    - A verificação é feita símbolo a símbolo

- $^*$ = inclui ε
- ω = ω (ômega) é uma string que pertence ao conjunto $T^*$
- Σ = alfabeto { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, . }
- L = linguagem { 0, 1, 2, ... , 99, 100, ... , 0.1, 0.2, ... }
- símbolo é um elemento individual do alfabeto
- palavra|cadeira|string = sequência de símbolos

# Analisador Léxico (AL)



- Análise léxica : segmentação do texto em palavras
  - Percorre-se o texto agrupando os símbolos em agregados (clusters) denominados itens léxicos:
    - Identificadores, nomes de variáveis, nomes de funções, etc.

- O tipo de gramática mais simples é a regular. Ex.:
  - $G = ( V , T , P , S )$, regra $A \rightarrow aB$, com $A, B \in V$, e $a, b \in T$
    - $A$ e $B$ : variáveis, $a$ e $b$ : terminais
      - Uma linguagem é regular se, e somente se, existir uma gramática regular que a gere

# Exemplo 1



- Linguagem L={aab, bba} sobre o alfabeto Σ={a, b}
  - A linguagem pode ser gerada pela gramática regular:
    - $S \rightarrow aA_1$ ,
    - $S \rightarrow bB_1$ ,
    - $A_1 \rightarrow aA_2$ ,
    - $A_2 \rightarrow b$ ,
    - $B_1 \rightarrow bB_2$ ,
    - $B_2 \rightarrow a$

símbolo terminal

A primeira cadeia pode ser gerada pela derivação:
$S \Rightarrow aA_1 \Rightarrow aaA_2 \Rightarrow aab$

A segunda cadeia:
$S \Rightarrow bB_1 \Rightarrow bbB_2 \Rightarrow bba$

| | | |
|---|---|---|
| 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 |

# LR e GR



- Dada uma gramática G , que não é regular, é possível que a linguagem L(G) seja regular, bastando que, para isso, exista uma gramática regular $G_2$ tal que $L(G_2) = L(G)$ . Ex.:
  - Considere a gramática não regular
    $S \rightarrow A_1b$ , $S \rightarrow B_1a$ , $A_1 \rightarrow A_2a$ , $A_2 \rightarrow a$ , $B_1 \rightarrow B_2b$ , $B_2 \rightarrow b$
    - $A_1 \rightarrow A_2a$ não é regular (para ser regular: $A \rightarrow aB$)
  - Como L(G) = {aab, bba} , visto anteriormente, e gerado por uma gramática regular, então L(G) é regular
    - $S \rightarrow A_1b \rightarrow A_2ab \rightarrow aab$
    - $S \rightarrow B_1a \rightarrow B_2ba \rightarrow bba$

conjunto de regras da gramática

ou seja, existe uma GR

Regras de produção para GR

$A \rightarrow aB$

$A \rightarrow a$

# Gramática linear à esquerda e direita



- Gramática linear à esquerda
  - $A \rightarrow b$ , $A \rightarrow \varepsilon$ , $A \rightarrow \boxed{Ba}$ , com $A, B \in V$ e $a, b \in T$
    - A linguagem gerada por gramáticas deste tipo é regular

- Gramáticas lineares à direita
  - $A \rightarrow b$ , $A \rightarrow \varepsilon$ , $A \rightarrow \boxed{aB}$ , com $A, B \in V$ e $a, b \in T$
    - Alguns autores consideram ambos os tipos gramáticas regulares,
      - tanto as lineares à esquerda quanto as lineares à direita

| | | |
|---|---|---|
| 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 |

# De GLC para GR

Regras de produção para GR
A → aB
A → a

- Regras A → B podem ser substituídas para obtermos uma gramática regular (GR) equivalente. Exemplo:
  - S → A | a
    A → aB | B
    B → b

    A → B não se encaixa na forma de uma GR

    derivações transitivas

    - Podemos observar que é possível $S \Rightarrow^* A$ , $S \Rightarrow^* B$ , $A \Rightarrow^* B$ :
      - $S \Rightarrow^* A$ : significa que S pode derivar A em *zero ou mais passos*, pois existe a produção S → A , em um único passo
      - $S \Rightarrow^* B$ : S pode derivar B em *zero ou mais passos* usando A (S ⇒ A) e, em seguida, A pode derivar B (A ⇒ B)
        - portanto, S pode derivar B em dois passos (S ⇒ A ⇒ B)
      - $A \Rightarrow^* B$ : olhando as regras, existe a produção A → B
        - portanto, em um único passo (A ⇒ B), A deriva B

(continua)

# De GLC para GR



- Podemos substituir o lado direito das regras simples da forma $A \rightarrow B$ pelo lado direito das regras cujo lado esquerdo é $B$
  - Regras com o lado esquerdo B: $B \rightarrow b$ (substituição indireta)
    - Substituimos o lado direito da regra $A \rightarrow B$ pelo lado direito da regra $B \rightarrow b$ , resultando $A \rightarrow b$

Antes da substituição:
$S \Rightarrow A \Rightarrow B \Rightarrow b$

Após a substituição:
$S \Rightarrow A \Rightarrow b$

- Gramática resultante
  - $S \rightarrow aB \mid b \mid a$
  - $A \rightarrow aB \mid b$
  - $B \rightarrow b$

era:
$S \rightarrow A \mid a$
$A \rightarrow aB \mid B$
$B \rightarrow b$

- A linguagem gerada por uma gramática apenas com regras regulares e regras simples, é regular

# LR e GR



- Se $L_1$ e $L_2$ são linguagens regulares, então $L_1 \cup L_2$ também é regular

- Se $L_1$ e $L_2$ são linguagens regulares, então existem gramáticas regulares $G_1 = ( V_1 , T , P_1 , S_1 )$ e $G_2 = ( V_2 , T , P_2 , S_2 )$ tais que $L(G_1) = L_1$ e $L(G_2) = L_2$
  - Podemos renomear as variáveis de $V_2$ de modo que não tenham o mesmo nome que uma variável de $V_1$
  - Podemos criar um novo símbolo inicial $S$ e criar a gramática

$$G_3 = ( V_1 \cup V_2 \cup \{S\}, T , P_1 \cup P_2 \cup \{ S \rightarrow S_1 , S \rightarrow S_2 \}, S )$$

  - $G_3$ possui regras regulares simples
  - $L(G_3) = L(G_1) \cup L(G_2)$, portanto, $L(G_1) \cup L(G_2)$ é regular

# Exemplo 2



- Sejam as gramáticas
  - $G_1$ : $S_1 \rightarrow bA_1$ , $A_1 \rightarrow a$
  - $G_2$ : $S_2 \rightarrow aA_2 \mid bB_2$ , $A_2 \rightarrow a$ , $B_2 \rightarrow bB_2 \mid b$
- Podemos construir a nova gramática $G_3$ tal que $L(G_3) = L(G_1) \cup L(G_2)$
  - $S \rightarrow S_1 \mid S_2$ , $S_1 \rightarrow bA_1$ , $A_1 \rightarrow a$ , $S_2 \rightarrow aA_2 \mid bB_2$ , $A_2 \rightarrow a$ , $B_2 \rightarrow bB_2 \mid b$
- $G_3$ é uma gramática apenas com regras regulares simples, e portanto, gera uma linguagem regular

# De $G_2$ para $L(G_2)^*$



- Desejamos uma gramática que gera a linguagem $L(G_2)^*$ ,
  - ou seja, o fecho de Kleene da linguagem gerada por $G_2$
    - É uma gramática que gera todas as strings que podem ser formadas pela concatenação de zero ou mais geradas por $G_2$

- $G_2$ : $S_2 \to aA_2$ , $S_2 \to bB_2$ , $A_2 \to a$ , $B_2 \to bB_2$ , $B_2 \to b$

- Em $G_2$ o símbolo inicial ($S_2$) não aparece do lado direito
  - Isso é importante pois para gerar o fecho de Kleene ($^*$), precisamos permitir a concatenação de strings geradas por $G_2$
  - Para isso, precisamos de uma forma de "voltar" para o símbolo inicial ($S_2$) após gerar uma string

# De $G_2$ para $L(G_2)^*$



- Como $G_2$ só tem regras na forma $A \rightarrow b$ e $A \rightarrow aB$ ,
  - basta colocar o símbolo inicial ($S_2$) nas regras, da forma $A \rightarrow b$
    - Isso permite que, após gerar um terminal, a gramática "retorne" ao estado inicial e gere outra string da linguagem
      - $A_2 \rightarrow a$ se torna $A_2 \rightarrow aS_2$ , e $B_2 \rightarrow b$ se torna $B_2 \rightarrow bS_2$
  - Acrescentar a regra $S_2 \rightarrow \varepsilon$
    - Adicionada para representar a concatenação de zero strings de $L(G_2)$, pois o fecho de Kleene ($^*$) inclui a string vazia

- Resultado final:

$$S_2 \rightarrow \varepsilon \text{ , } S_2 \rightarrow aA_2 \text{ , } S_2 \rightarrow bB_2 \text{ , } A_2 \rightarrow aS_2 \text{ , } B_2 \rightarrow bB_2 \text{ , } B_2 \rightarrow bS_2$$

# Exemplo 3



- Gramática, com símbolo inicial $A_0$ , gera os números inteiros, escritos na base 2, que deixam resto 1 quando divididos por 3 :
  - $A_0 \rightarrow 0A_0 \mid 1A_1$
  - $A_1 \rightarrow 0A_2 \mid 1A_0 \mid \varepsilon$
  - $A_2 \rightarrow 0A_1 \mid 1A_2$

$A_0$ : símbolo inicial
$A_0$, $A_1$, $A_2$ : variáveis
0, 1 : terminais

- Observe que $A_0 \Rightarrow^* wA_i$ se, e somente se, w é um número na base 2 que deixa resto i quando dividido por 3
  - Se a derivação a partir de $A_0$ termina em $A_0$, então a string binária gerada (w) representa um número que, quando dividido por 3, tem resto 0
  - Se a derivação a partir de $A_0$ termina em $A_1$, então a string binária gerada (w) representa um número que, quando dividido por 3, tem resto 1
  - Se a derivação a partir de $A_0$ termina em $A_2$, então a string binária gerada (w) representa um número que, quando dividido por 3, tem resto 2

(continua)

- Gerando o número binário 1 (decimal 1)
  - $A_0 \Rightarrow 1A_1 \Rightarrow 1\varepsilon = 1$
    - Como a derivação termina em $A_1$, o número 1 (decimal) deixa resto 1 quando dividido por 3

- Gerando o número binário 100 (decimal 4)
  - $A_0 \Rightarrow 1A_1 \Rightarrow 10A_2 \Rightarrow 100A_1 \Rightarrow 100\varepsilon = 100$
  - Como a derivação termina em $A_1$, o número 4 (decimal) deixa resto 1 se dividido por 3

(continua)

- Gerando o número binário 111 (decimal 7)
  - $A_0 \Rightarrow 1A_1 \Rightarrow 11A_0 \Rightarrow 110A_0 \Rightarrow 1100A_0 \Rightarrow 11000A_0 \ldots$
    - podemos continuar adicionando zeros indefinidamente, e o resto continuará sendo 1
  - $A_0 \Rightarrow 1A_1 \Rightarrow 11A_0 \Rightarrow 111A_1 \Rightarrow 111\varepsilon = 111$
    - como a derivação termina em $A_1$, o número 7 (decimal) deixa resto 1 quando dividido por 3
- Gerando o número binário 10 (decimal 2)
  - $A_0 \Rightarrow 1A_1 \Rightarrow 10A_2 \Rightarrow 10\varepsilon = 10$
    - como a derivação termina em $A_2$, o número 2 (decimal) deixa resto 2 quando dividido por 3

(continua)

- Como a gramática "calcula" o resto:
  - A gramática implementa uma espécie de autômato finito que simula a divisão por 3
  - A cada novo bit lido (0 ou 1), o estado ($A_0$, $A_1$ ou $A_2$) muda, representando o novo resto
    - Começamos em $A_0$ (resto 0)
    - Ler um '0' não altera o resto ($A_0 \to 0A_0$, $A_1 \to 0A_2$, $A_2 \to 0A_1$)
    - Ler um '1' incrementa o resto em 1 ($A_0 \to 1A_1$, $A_1 \to 1A_0$, $A_2 \to 1A_2$)
  - Como queremos gerar números com resto 1, as derivações devem terminar em $A_1$
    - A regra $A_1 \to \varepsilon$ permite que a derivação termine e, assim, gere a string binária

(continua)

# Exemplo 4



- Gramática regular para ler comentários em Python:

  A → "#" $B^*$

  B → [a-zA-Z0-9_-+*/=.,!@#$%^&*{}:;<>?~\ ]

- Onde:
  - "#" representa o caractere de início do comentário
  - $B^*$ representa zero ou mais ocorrências de caracteres válidos