Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

Departamento de Informática Teórica

Instituto de Informática / UFRGS

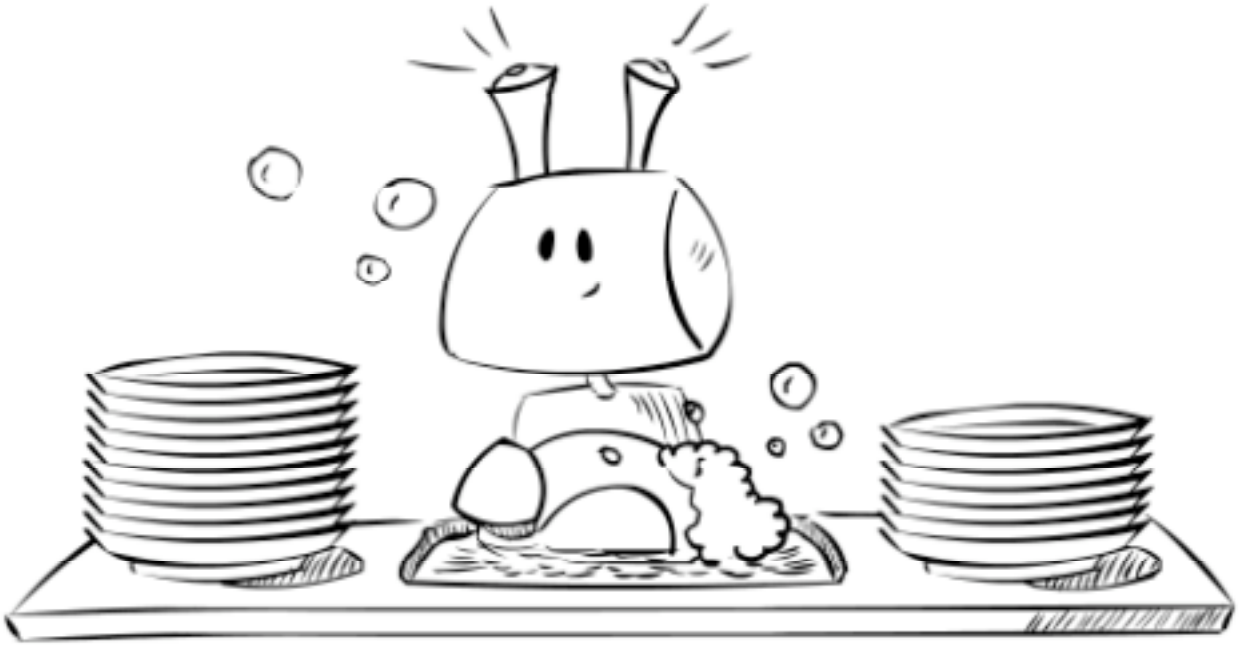
Linguagens Formais e Autômatos

P. Blauth Menezes

1 Introdução e Conceitos Básicos
2 Linguagens e Gramáticas
3 Linguagens Regulares
4 Propriedades das Linguagens Regulares
5 Autômato Finito com Saída
6 Linguagens Livres do Contexto
7 Propriedades e Reconhecimento das Linguagens Livres do Contexto
8 Linguagens Recursivamente Enumeráveis e Sensíveis ao Contexto
9 Hierarquia de Classes e Linguagens e Conclusões

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha



6 – Linguagens Livres do Contexto

6 Linguagens Livres do Contexto

- **Estudo da Classe das Linguagens Livres do Contexto ou Tipo 2 é de fundamental importância**
  - universo mais amplo de linguagens comparativamente com as LR
  - trata, adequadamente, questões típicas de linguagens de programação
    - parênteses balanceados
    - construções bloco-estruturadas, etc.
- **Algoritmos reconhecedores e geradores**
  - relativamente simples
  - eficiência razoável

Aplicações típicas

- centradas em linguagens artificiais
  - em especial, nas linguagens de programação
- analisadores sintáticos
- tradutores de linguagens
- processadores de texto em geral

Hierarquia de Chomsky

- Classe das Linguagens Livres do Contexto
- contém propriamente a Classe das Linguagens Regulares

Entretanto, é uma classe relativamente restrita

- fácil definir linguagens que não pertencem a esta classe

◆ Abordagens

- Gramática Livre do Contexto (axiomático ou gerador)
  - restrições na forma das regras de produção
  - mais livre que na gramática regular

- Autômato com Pilha (operacional ou reconhecedor)
  - análogo ao autômato finito não-determinístico
  - adicionalmente: memória auxiliar tipo pilha
  - pode ser lida ou gravada

Relativamente às GLC

- Árvore de derivação
  - representa a derivação de uma palavra na forma de árvore
  - parte do símbolo inicial como a raiz
  - termina em símbolos terminais como folhas
- Gramática Ambígua
  - pelo menos uma palavra com duas ou mais árvores de derivação
- Simplificação de Gramática (produções)
  - sem reduzir o poder de geração
- Forma Normal: restrições rígidas na forma das produções
  - sem reduzir o poder de geração da gramática

◆ Autômato com pilha construído a partir de uma GLC

- construção de um reconhecedor a partir de sua gramática
  - simples e imediata

- estrutura de pilha é suficiente como única memória
  - pode ser reconhecida por autômato com pilha com um estado
  - estados não são necessários para "memorizar" o passado

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.1 Gramática Livre do Contexto

Def: Gramática Livre do Contexto (GLC)

$$G = (V, T, P, S)$$

qualquer regra de produção é da forma

$$A \rightarrow \alpha$$

- A é variável de V
- α é palavra de $(V \cup T)^*$

lado esquerdo = uma variável

Def: Linguagem Livre do Contexto (LLC) ou Tipo 2

Linguagem gerada pela gramática livre do contexto G

$$GERA(G) = \{ w \in T^* \mid S \Rightarrow^+ w \}$$

- **Portanto, é livre do contexto**
  - qualquer linguagem regular

◆ Relação entre as classes de linguagens estudadas

◆ "Livre do contexto" ???

- mais geral classe de linguagens cuja produção é da forma $A \rightarrow \alpha$
- em uma derivação, a variável A deriva α
  - sem depender ("livre") de qualquer análise
  - dos símbolos que antecedem ou sucedem A (o "contexto")
  - na palavra que está sendo derivada

Exp: GLC: Duplo Balanceamento

$$L_1 = \{ a^n b^n \mid n \geq 0 \}$$

$G_1 = (\{ S \}, \{ a, b \}, P_1, S)$

- $P_1 = \{ S \rightarrow aSb \mid S \rightarrow \varepsilon \}$
- $GERA(G_1) = L_1$

Derivação da palavra aabb

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\varepsilon bb = aabb$$

◆ Importante: Duplo Balanceamento

- analogia com estruturas de duplo balanceamento
  - em linguagens de programação
- linguagens bloco-estruturadas
  - $\mathbf{begin}^n\,\mathbf{end}^n$ e similares
- linguagens com parênteses balanceados
  - $(^n\,)^n$

Exp: GLC: Expressões Aritméticas

L_2 - expressões aritméticas com colchetes balanceados, dois operadores e um operando

$$G_2 = (\{ E \}, \{ +, *, [,], x \}, P_2, E)$$

- $P_2 = \{ E \rightarrow E+E \mid E*E \mid [E] \mid x \}$

Derivação da expressão $[x+x]*x$

$$E \Rightarrow E*E \Rightarrow [E]*E \Rightarrow [E+E]*E \Rightarrow [x+E]*E \Rightarrow [x+x]*E \Rightarrow [x+x]*x$$

- existe outra seqüência de derivação? Quantas?
- quais produções controlam o duplo balanceamento de parênteses?

Obs: BNF: *Backus Naur Form*

Maneira usual de representar uma GLC

- variáveis
  - palavras delimitadas pelos símbolos $\langle$ e $\rangle$
- terminais
  - palavras não-delimitadas
- representação de uma regra de produção $A \rightarrow \alpha$

$$A ::= \alpha$$

Exp: BNF: Identificador em Pascal

A variável ⟨identificador⟩ é o símbolo inicial

- ⟨identificador⟩ ::=
  ⟨letra⟩ | ⟨identificador⟩⟨letra⟩ | ⟨identificador⟩⟨dígito⟩
- ⟨letra⟩ ::= a | b | ... | z
- ⟨dígito⟩ ::= 0 | 1 | ... | 9

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.2 Árvore de Derivação

- **Derivação de palavras na forma de árvore**
  - partindo do símbolo inicial como a raiz
  - terminando em símbolos terminais como folhas
- **Conveniente em muitas aplicações**
  - Compiladores
  - processadores de textos

Def: Árvore de Derivação

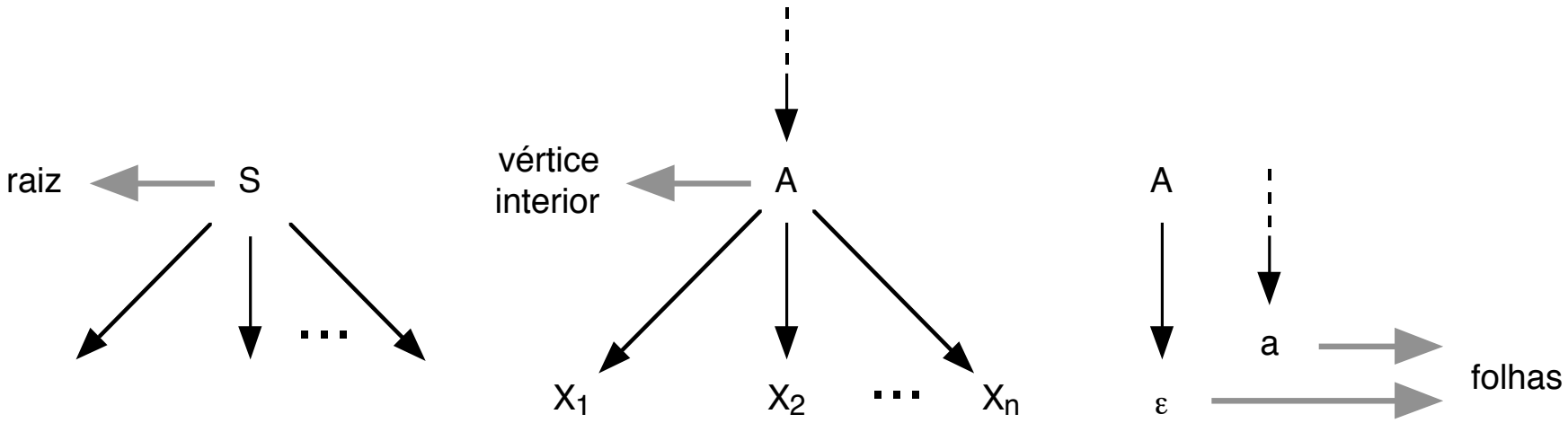
Raiz: símbolo inicial

Vértices interiores: variáveis

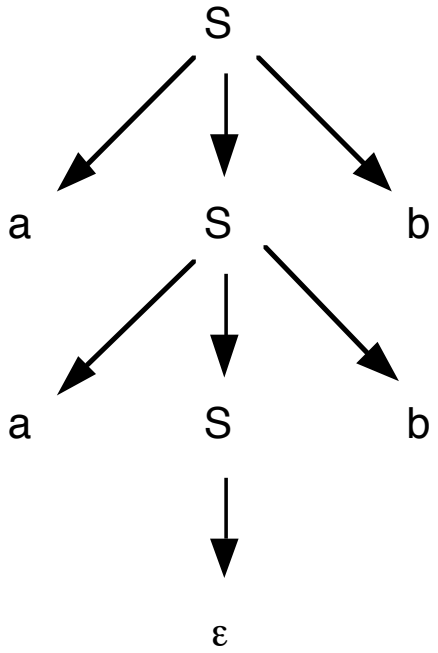
- se A é um vértice interior e $X_1, X_2, \ldots, X_n$ são os "filhos" de A
  - $A \rightarrow X_1 X_2 \ldots X_n$ é uma produção da gramática
  - $X_1, X_2, \ldots, X_n$ são ordenados da esquerda para a direita

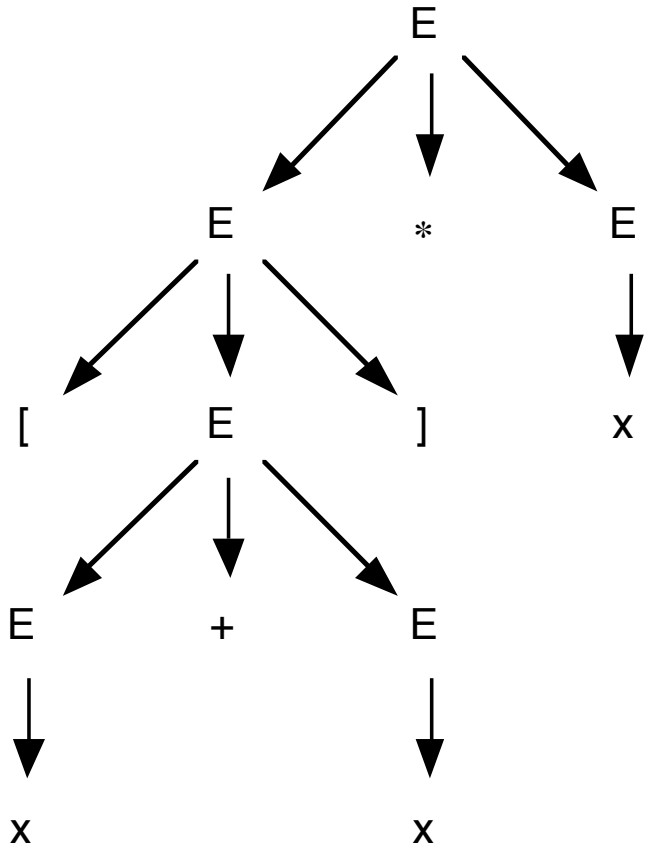
Vértice folha ou folha: terminal ou o símbolo vazio

- se vazio: único filho de seu pai ($A \rightarrow \varepsilon$)



Exp: Árvore de Derivação: aabb e [x+x]∗x

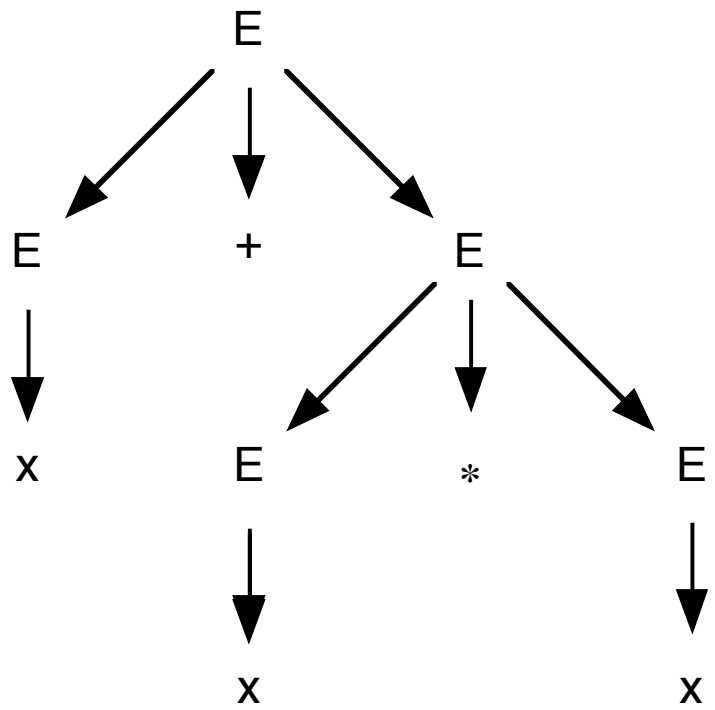




- **Uma árvore de derivação**
  - pode representar derivações distintas de uma mesma palavra

Exp: Árvore de Derivação × Derivações: x+x*x

- $E \Rightarrow E+E \Rightarrow x+E \Rightarrow x+E*E \Rightarrow x+x*E \Rightarrow x+x*x$ mais a esquerda
- $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+E*x \Rightarrow E+x*x \Rightarrow x+x*x$ mais a direita
- $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow x+E*E \Rightarrow x+x*E \Rightarrow x+x*x$
- etc…

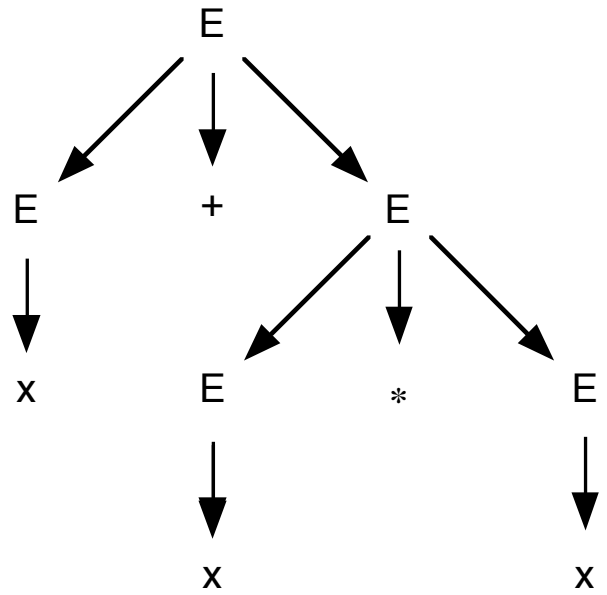


Def: Derivação mais à Esquerda (Direita)

Seqüência de produções aplicada sempre à variável mais à esquerda (direita) da palavra

Exp: Derivação mais à Esquerda (Direita): x+x∗x

- $E \Rightarrow E+E \Rightarrow x+E \Rightarrow x+E*E \Rightarrow x+x*E \Rightarrow x+x*x$ mais a esquerda
- $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+E*x \Rightarrow E+x*x \Rightarrow x+x*x$ mais a direita



6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

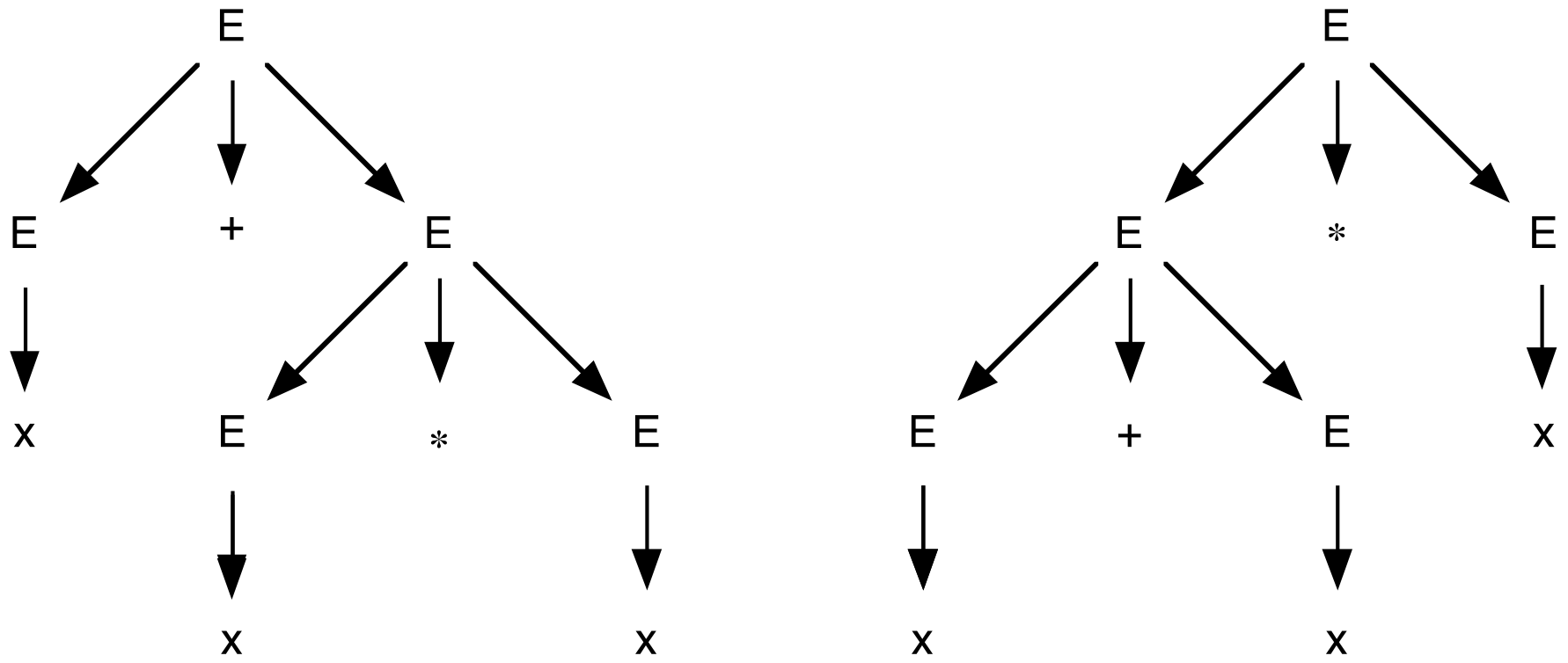
6.3 GLC Ambígua

- **Gramática ambígua**
  - uma palavra associada a duas ou mais árvores de derivação
- **Pode ser desejável que a gramática usada seja não-ambígua**
  - desenvolvimento e otimização de alguns algoritmos de reconhecimento
- **Nem sempre é possível eliminar ambigüidades**
  - é fácil definir linguagens para as quais qualquer GLC é ambígua

Def: Gramática (Livre do Contexto) Ambígua

Existe pelo menos uma palavra que possui duas ou mais árvores de derivação

Exp: Gramática Ambígua: x+x*x



◆ Mais de uma derivação à esquerda (direita)

x+x*x

- Derivação mais à esquerda
  - E ⇒ E+E ⇒ x+E ⇒ x+E*E ⇒ x+x*E ⇒ x+x*x
  - E ⇒ E*E ⇒ E+E*E ⇒ x+E*E ⇒ x+x*E ⇒ x+x*x
- Derivação mais à direita
  - E ⇒ E+E ⇒ E+E*E ⇒ E+E*x ⇒ E+x*x ⇒ x+x*x
  - E ⇒ E*E ⇒ E*x ⇒ E+E*x ⇒ E+x*x ⇒ x+x*x

◆ Forma equivalente de definir gramática ambígua

- existe pelo menos uma palavra com duas ou mais derivações mais à esquerda
  - alternativamente, mais à direita

Teorema: Gramática Ambígua

Uma GLC é uma Gramática Ambígua se existe pelo menos uma palavra

- duas ou mais derivações mais à esquerda ou
- duas ou mais derivações mais à direita

Def: Linguagem Inerentemente Ambígua

Qualquer GLC é ambígua

Exp: Linguagem Inerentemente Ambígua

$$\{ w \mid w = a^n b^n c^m d^m \text{ ou } w = a^n b^m c^m d^n, n \geq 1, m \geq 1 \}$$

Exp: Linguagem Inerentemente Ambígua: contra-exemplo

Expressões aritméticas é não-ambígua (exercício)

Correto entendimento da solução é especialmente importante

- justifica a maneira aparentemente "estranha" de definir expressões
- na maioria das gramáticas das linguagens de programação

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.4.1 Símbolos Inúteis
6.4.2 Produções Vazias
6.4.3 Produções que Substituem Variáveis
6.4.4 Simplificações Combinadas
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.4 Simplificação de GLC

- **Simplificação de alguns tipos de produções**
  - sem reduzir o poder de geração das GLC
- **Simplificações são importantes**
  - construção e otimização de algoritmos
  - demonstração de teoremas

◆ Simplificações

- Símbolos inúteis
  - exclusão de variáveis ou terminais não-usados
- Produções vazias, da forma $A \rightarrow \varepsilon$
  - se ε pertence à linguagem: incluída produção vazia específica
- Produções que substituem variáveis, da forma $A \rightarrow B$
  - substituem uma variável por outra
  - não adicionam informação de geração de palavras

◆ Provas omitidas

- algoritmos de simplificação atingem os objetivos propostos

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.4.1 Símbolos Inúteis
6.4.2 Produções Vazias
6.4.3 Produções que Substituem Variáveis
6.4.4 Simplificações Combinadas
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.4.1 Símbolos Inúteis

- **Símbolos inúteis**
  - símbolos não-usados na geração de palavras de terminais
- **Simplificação exclui**
  - produções que fazem referência a esses símbolos
  - os próprios símbolos inúteis
  - não é necessária qualquer modificação adicional

◆ Algoritmo

- *Etapa 1: qualquer variável gera terminais*

  restringe o conjunto de variáveis

  * considera todas as variáveis que geram terminais diretamente (exemplo: A → a)
  * adiciona, sucessivamente, variáveis que geram terminais indiretamente (exemplo: B → Ab)

- *Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial*

  analisa as produções da gramática a partir do símbolo inicial

  * considera exclusivamente o símbolo inicial
  * sucessivamente as produções da gramática são aplicadas: símbolos referenciados são adicionados aos novos conjuntos

Def: Algoritmo: Exclusão dos Símbolos Inúteis

$$G = (V, T, P, S)$$ GLC

Etapa 1: qualquer variável gera terminais. Gramática resultante

$$G_1 = (V_1, T, P_1, S)$$

- construção de $V_1 \subseteq V$

$V_1 = \varnothing$;
repita $V_1 = V_1 \cup \{ A \mid A \rightarrow \alpha \in P \text{ e } \alpha \in (T \cup V_1)^* \}$
até que o cardinal de V_1 não aumente;

- P_1 possui os mesmos elementos que P, excetuando-se
  - produções cujas variáveis não pertencem a V_1

Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial

Gramática resultante

$$G_2 = (V_2, T_2, P_2, S)$$

```
T2 = ∅;
V2 = { S };
repita
    V2 = V2 ∪ { A | X → α A β ∈ P1, X ∈ V2 };
    T2 = T2 ∪ { a | X → α a β ∈ P1, X ∈ V2 }
até que os cardinais de V2 e T2 não aumentem;
```

- P_2 possui os mesmos elementos que P_1, excetuando-se
  - produções cujos símbolos não pertencem a V_2 ou T_2

- **Se as etapas forem executadas em ordem inversa (*Etapa 2* antes da *Etapa 1*)**
  - pode *não* atingir o resultado esperado
  - demonstração: apresentar um contra-exemplo (exercício)

Exp: Exclusão dos Símbolos Inúteis

$$G = (\{ S, A, B, C \}, \{ a, b, c \}, P, S)$$

- $P = \{ S \rightarrow aAa \mid bBb, A \rightarrow a \mid S, C \rightarrow c \}$

Etapa 1: qualquer variável gera terminais

Iteração	Variáveis
início	$\varnothing$
1	$\{ A, C \}$
2	$\{ A, C, S \}$
3	$\{ A, C, S \}$

- $S \rightarrow bBb$ é excluída: B não pertence ao novo conjunto de variáveis

- gramática resultante da *etapa 1*

$$G_1 = (\{ A, C, S \}, \{ a \}, \{ S \rightarrow aAa, A \rightarrow a \mid S, C \rightarrow c \}, S)$$

Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial

Iteração	Variáveis	Terminais
início	$\{ S \}$	$\varnothing$
1	$\{ S, A \}$	$\{ a \}$
2	$\{ S, A \}$	$\{ a \}$

- $C \rightarrow c$ é excluída: C e c não pertencem aos novos conjuntos
- gramática resultante da *etapa 2*

$$G_2 = (\{ S, A \}, \{ a \}, \{ S \rightarrow aAa, A \rightarrow a \mid S \}, S)$$

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.4.1 Símbolos Inúteis
6.4.2 Produções Vazias
6.4.3 Produções que Substituem Variáveis
6.4.4 Simplificações Combinadas
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.4.2 Produções Vazias

- **Exclusão de produções vazias (da forma $A \rightarrow \varepsilon$)**
  - pode determinar modificações diversas nas produções
- **Algoritmo**
  - *Etapa 1: variáveis que constituem produções vazias*
    - $A \rightarrow \varepsilon$: variáveis que geram diretamente ε
    - $B \rightarrow A$: sucessivamente, variáveis que indiretamente geram ε
  - *Etapa 2: exclusão de produções vazias*
    - considera apenas as produções não-vazias
    - cada produção cujo lado direito possui uma variável que gera ε, determina uma produção adicional, sem essa variável
  - *Etapa 3: geração da palavra vazia, se necessário*

Def: Algoritmo: Exclusão das Produções Vazias

$G = (V, T, P, S)$ GLC

Etapa 1: variáveis que constituem produções vazias

- V_ε, conjunto das variáveis que geram ε

$V_\varepsilon = \{ A \mid A \rightarrow \varepsilon \}$;
repita
    $V_\varepsilon = V_\varepsilon \cup \{ X \mid X \rightarrow X_1 \ldots X_n \in P$
    tal que $X_1, \ldots, X_n \in V_\varepsilon \}$
até que o cardinal de V_ε não aumente;

Etapa 2: exclusão de produções vazias

Gramática resultante

$$G_1 = (V, T, P_1, S)$$

- construção de P_1

```
P1 = { A → α | α ≠ ε };
repita
    para   toda A → α ∈ P1, X ∈ Vε tal que
           α = α1 X α2, α1 α2 ≠ ε
    faça   P1 = P1 ∪ { A → α1 α2 }
até que o cardinal de P1 não aumente;
```

Etapa 3: geração da palavra vazia, se necessário

- se ε pertence à linguagem
  - introduz a produção $S \rightarrow \varepsilon$
- gramática resultante

$$G_2 = (V, T, P_2, S)$$

  - $P_2 = P_1 \cup \{ S \rightarrow \varepsilon \}$

Exp: Exclusão das Produções Vazias

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

- $P = \{ S \rightarrow aXa \mid bXb \mid \varepsilon,\ X \rightarrow a \mid b \mid Y,\ Y \rightarrow \varepsilon \}$

Etapa 1: variáveis que constituem produções vazia

Iteração	V_ε
início	{ S, Y }
1	{ S, Y, X }
2	{ S, Y, X }

Etapa 2: exclusão de produções vazias

Iteração	Produções
início	$\{ S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid Y \}$
1	$\{ S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y \}$
2	$\{ S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y \}$

- Gramática resultante

$$G_1 = (\{ S, X, Y \}, \{ a, b \}, \{ S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y \}, S)$$

Etapa 3: geração da palavra vazia, se necessário.

- palavra vazia pertence à linguagem: $S \rightarrow \varepsilon$ é incluída

Gramática resultante

$$G_2 = (\{ S, X, Y \}, P_2, S)$$

- $P_2 = \{ a, b \}, \{ S \rightarrow aXa \mid bXb \mid aa \mid bb \mid \varepsilon,\ X \rightarrow a \mid b \mid Y \}$

◆ Observe

- Y, originalmente um símbolo útil, resultou em um símbolo inútil
- exclusão de produções vazias gerou símbolo inútil

◆ Conclusão

não é qualquer combinação de simplificações de gramática

que atinge o resultado desejado

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.4.1 Símbolos Inúteis
6.4.2 Produções Vazias
6.4.3 Produções que Substituem Variáveis
6.4.4 Simplificações Combinadas
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.4.3 Produções que Substituem Variáveis

- **Produção que substitui uma variável por outra**
  - tipo $A \rightarrow B$
    - não adiciona informação em termos de geração de palavras
  - se $B \rightarrow \alpha$, então
    - $A \rightarrow B$ pode ser substituída por $A \rightarrow \alpha$
  - generalização da idéia: algoritmo proposto

◆ Algoritmo

- *Etapa 1: fecho transitivo de cada variável*
  - conjunto de variáveis que podem substituí-la transitivamente
  - ex: se $A \rightarrow B$ e $B \rightarrow C$, então B e C pertencem ao fecho de A

- *Etapa 2: exclusão das produções que substituem variáveis*
  - se α é atingível a partir de A através de seu fecho
  - substitui $A \rightarrow B$ por $A \rightarrow \alpha$

Def: Algoritmo: Exclusão das Produções que Substituem Variáveis

$G = (V, T, P, S)$ GLC

Etapa 1: fecho transitivo de cada variável

para toda $A \in V$
faça FECHO-A = $\{ B \mid A \neq B$ e $A \Rightarrow^{+} B$ usando
exclusivamente produções de P da forma $X \rightarrow Y \}$;

Etapa 2: exclusão das produções que substituem variáveis

Gramática resultante

$$G_1 = (V, T, P_1, S)$$

- construção de P_1

```
P1 = { A → α | A → α ∈ P e α ∉ V };
 para   toda A ∈ V e B ∈ FECHO-A
 faça   se  B → α ∈ P e α ∉ V
        então  P1 = P1 ∪ { A → α };
```

Exp: Exclusão das Produções que Substituem Variáveis

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

- $P = \{ S \rightarrow aXa \mid bXb,\ X \rightarrow a \mid b \mid S \mid \varepsilon \}$

Etapa 1: fecho transitivo da cada variável

- FECHO-S = $\varnothing$
- FECHO-X = $\{ S \}$

Etapa 2: exclusão das produções da forma $A \rightarrow B$

Iteração	Produções
inicial	$\{ S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \}$
S	$\{ S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \}$
X	$\{ S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \mid aXa \mid bXb \}$

Gramática resultante

$$G_1 = (\{ S, X \}, \{ a, b \}, P_1, S)$$

- $P_1 = \{ S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \mid aXa \mid bXb \}, S)$

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.4.1 Símbolos Inúteis
6.4.2 Produções Vazias
6.4.3 Produções que Substituem Variáveis
6.4.4 Simplificações Combinadas
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.4.4 Simplificações Combinadas

- ***Não*** **é qualquer combinação de simplificações de GLC**
  - que atinge o resultado desejado
- **Exemplo: gramática *sem* símbolos inúteis, mas *com* produções que substituem variáveis**
  - algoritmo para excluir produções que substituem variáveis pode gerar símbolos inúteis (por quê?)
- **Seqüência de simplificação recomendada**
  - *Exclusão das produções vazias*
  - *Exclusão das produções que substituem variáveis*
  - *Exclusão dos símbolos inúteis*

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.5.1 Forma Normal de Chomsky
6.5.2 Forma Normal de Greibach
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.5 Formas Normais

- **Formas normais**
  - restrições rígidas na forma das produções
  - sem reduzir o poder de geração das GLC
    - excetuando-se a geração da palavra vazia
- **Aplicações**
  - desenvolvimento de algoritmos
    - destaque para reconhecedores de linguagens
  - prova de teoremas

- **Forma Normal de Chomsky: produções são da forma**

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a$$

- **Forma Normal de Greibach: produções são da forma**

$$A \rightarrow a\alpha \qquad \alpha \text{ palavra de } \textit{variáveis}$$

- **Algoritmos de conversão**
  - provas omitidas
  - de que os algoritmos atingem os objetivos propostos

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
    6.5.1 Forma Normal de Chomsky
    6.5.2 Forma Normal de Greibach
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.5.1 Forma Normal de Chomsky

Def: Forma Normal de Chomsky (FNC)

$G = (V, T, P, S)$ GLC

Todas produções são da forma (A, B e C são variáveis, a é terminal)

$A \rightarrow BC$ ou $A \rightarrow a$

- **Palavra vazia**
  - não pertence à linguagem gerada por uma gramática na FNC

◆ Algoritmo: três etapas

- *Etapa 1: simplificação da gramática*
  - $A \rightarrow \varepsilon$ — linguagem não possui ε
  - $A \rightarrow B$ — um símbolo no lado direito: terminal
  - símbolos inúteis — opcional
- *Etapa 2: variáveis no lado direito das produções*
  - lado direito de comprimento ≥ 2: exclusivamente variáveis
  - se for um terminal?
- *Etapa 3: exatamente duas variáveis no lado direito das produções*
  - como transformar produções da forma $A \rightarrow B_1 B_2 \ldots B_n$ $(n \geq 2)$?

Def: Algoritmo - Forma Normal de Chomsky

$G = (V, T, P, S)$ GLC tal que $\varepsilon \notin$ GERA(G)

Etapa 1: simplificação da gramática

$G_1 = (V_1, T_1, P_1, S)$ gramática resultante

- simplificações combinadas (algoritmos estudados)
  - produções vazias
  - produções que substituem variáveis
  - símbolos inúteis (opcional)

Etapa 2: transformação do lado direito das produções de comprimento maior ou igual a dois

$G_2 = (V_2, T_1, P_2, S)$ gramática resultante

- construção de V_2 e P_2 (para cada variável a, suponha $C_a \notin V_2$)

```
V2 = V1;
P2 = P1;
para   toda A → X1X2...Xn ∈ P2 tal que n ≥ 2
faça   se para r ∈ { 1,...,n }, Xr é um símbolo terminal
       então (suponha Xr = a)
             V2 = V2 ∪ { Ca };
             substitui a por Ca em A → X1X2...Xn ∈ P2;
             P2 = P2 ∪ { Ca → a };
```

Etapa 3: transformação do lado direito das produções de comprimento maior ou igual a três em produções com exatamente duas variáveis

$G_3 = (V_3, T_1, P_3, S)$ gramática resultante

- construção de V_3 e P_3
  - a cada ciclo, suponha $D_1 \notin V_3, \ldots, D_{n-2} \notin V_3$)

$V_3 = V_2$;
$P_3 = P_2$;
para toda $A \rightarrow B_1B_2\ldots B_n \in P_3$ tal que $n \geq 3$
faça $P_3 = P_3 - \{ A \rightarrow B_1B_2\ldots B_n \}$;
$V_3 = V_3 \cup \{ D_1,\ldots,D_{n-2} \}$;
$P_3 = P_3 \cup \{ A \rightarrow B_1D_1,\ D_1 \rightarrow B_2D_2,\ldots,$
$D_{n-3} \rightarrow B_{n-2}D_{n-2},\ D_{n-2} \rightarrow B_{n-1}B_n \}$;

Exp: Algoritmo: Forma Normal de Chomsky

$G = (\{ E \}, \{ +, *, [,], x \}, P, E)$ expr. aritméticas

- $P = \{ E \rightarrow E+E \mid E*E \mid [E] \mid x \}$

Etapa 1: simplificação da gramática já está simplificada

Etapa 2: lado direito das produções de comprimento ≥ 2

- $E \rightarrow x$ está OK
- demais produções
  - $E \rightarrow E\, C_+\, E \mid E\, C_*\, E \mid C_[\, E\, C_]$
  - $C_+ \rightarrow +$
  - $C_* \rightarrow *$
  - $C_[\rightarrow [$
  - $C_] \rightarrow]$

Etapa 3: exatamente duas variáveis no lado direito das produções

- produções

$$E \rightarrow E\, C_{+}\, E \mid E\, C_{*}\, E \mid C_{[}\, E\, C_{]}$$

- substituídas por
  - $E \rightarrow E\, D_1 \mid E\, D_2 \mid C_{[}\, D_3$
  - $D_1 \rightarrow C_{+}\, E$
  - $D_2 \rightarrow C_{*}\, E$
  - $D_3 \rightarrow E\, C_{]}$

Gramática resultante, na Forma Normal de Chomsky

$$G_{FNC} = (\{ E, C_+, C_*, C_[, C_], D_1, D_2, D_3 \}, \{ +, *, [,], x \}, P_{FNC}, E)$$

- Produções de P_{FNC}
  - $E \rightarrow E\ D_1 \mid E\ D_2 \mid C_[\ D_3 \mid x$,
  - $D_1 \rightarrow C_+\ E$
  - $D_2 \rightarrow C_*\ E$
  - $D_3 \rightarrow E\ C_]$,
  - $C_+ \rightarrow +$
  - $C_* \rightarrow *$
  - $C_[\rightarrow [$
  - $C_] \rightarrow]$

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
    6.5.1 Forma Normal de Chomsky
    6.5.2 Forma Normal de Greibach
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.5.2 Forma Normal de Greibach

Def: Forma Normal de Greibach (FNG)

$G = (V, T, P, S)$ GLC

Todas as suas produções são da forma (α é uma palavra de V^*)

$$A \rightarrow a\alpha$$

- **Palavra vazia**
  - não pertence à linguagem gerada por uma gramática na FNG

◆ Algoritmo (etapas)

Etapa 1: simplificação da gramática. Análoga à FNC

- $A \rightarrow \varepsilon$ — linguagem não possui ε
- $A \rightarrow B$ — primeiro símbolo no lado direito: terminal
- símbolos inúteis — opcional

Etapa 2: renomeação das variáveis em uma ordem crescente

- exemplo: $A_1, A_2, \ldots, A_n$ — $\#V = n$
- diferentes critérios de renomeação
  - diferentes gramáticas na FNG
  - todas equivalentes (geram a mesma linguagem)

Etapa 3: produções na forma $A_r \rightarrow A_s\alpha$, em que $r \leq s$

- $A_r \rightarrow A_s\alpha$ tais que $r > s$ são modificadas
  - substitui A_s pelas suas produções ($A_s \rightarrow \beta_1 \mid \ldots \mid \beta_m$)
  - resulta em $A_r \rightarrow \beta_1\alpha \mid \ldots \mid \beta_m\alpha$
  - e assim sucessivamente

- conjunto de variáveis é finito: limite para produções crescentes
  - geração de terminal $A_r \rightarrow a\alpha$
  - geração de recursão $A_r \rightarrow A_r\alpha$

Etapa 4: exclusão das recursões da forma $A_r \rightarrow A_r\alpha$

- podem existir originalmente na gramática
- ou serem geradas pela etapa anterior
- eliminação da recursão à esquerda
  - introduz variáveis auxiliares
  - inclui recursão à direita

$B_r \rightarrow \alpha B_r$

Etapa 5: um terminal no início do lado direito de cada produção

- produções da forma $A_r \rightarrow A_s\alpha$ são tais que $r < s$
- portanto, produções da maior variável A_n
  - obrigatoriamente iniciam por terminal no lado direito
- $A_{n-1} \rightarrow A_n\alpha$: substituí A_n pelas suas produções ($A_n \rightarrow a\beta$)
  - lado direito das produções de A_{n-1} também inicia por terminal
  - exemplo: $A_{n-1} \rightarrow a\beta\alpha$
- repetição para $A_{n-2},\ldots,A_1$
  - resulta em produções exclusivamente da forma $A_r \rightarrow a\alpha$

Etapa 6: produções na forma $A \rightarrow a\alpha$, α palavra de variáveis

- análoga à correspondente etapa do algoritmo relativo à FNC

Def: Algoritmo: Forma Normal de Greibach

$G = (V, T, P, S)$ GLC tq $\varepsilon \notin$ GERA(G)

Etapa 1: simplificação da gramática

$G_1 = (V_1, T_1, P_1, S)$ gramática resultante

- simplificações combinadas (algoritmos estudados)
  - produções vazias
  - produções que substituem variáveis
  - símbolos inúteis (opcional)

Etapa 2: renomeação das variáveis em uma ordem crescente qualquer.

$G_2 = (V_2, T_1, P_2, A_i)$ gramática resultante

- suponha que A_i corresponde à renomeação de S

Etapas 3 e 4: transformação de produções para a forma $A_r \rightarrow A_s\alpha$, *na qual* $r \leq s$ e *exclusão das recursões da forma* $Ar \rightarrow A_r\alpha$

$G_3 = (V_3, T_1, P_3, A_i)$ gramática resultante

- construção de V_3 e P_3 supondo
  - cardinal de V_2 é n
  - a cada ciclo, $B_r \notin V_3$

```
P3 = P2
para   r variando de 1 até n
faça
    para   s variando de 1 até r-1                    Etapa 3
    faça   para   toda Ar → As α ∈ P3
           faça   excluir Ar → As α de P3;
                  para toda As → β ∈ P3
                  faça P3 = P3 ∪ { Ar → β α }
    para   toda Ar → Ar α ∈ P3                        Etapa 4
    faça   excluir Ar → Ar α de P3;
           V3 = V3 ∪ { Br };
           P3 = P3 ∪ { Br → α } ∪ { Br → α Br };
    para   toda Ar → ϕ ∈ P3 tq ϕ não inicia por Ar e
           alguma Ar → Ar α foi excluída
    faça   P3 = P3 ∪ { Ar → ϕ Br };
```

Etapa 5: um terminal no início do lado direito de cada produção

$G_4 = (V_3, T_1, P_4, A_i)$ gramática resultante

- construção de P_4

$P_4 = P_3$;
para r variando de n-1 até 1 e toda $A_r \rightarrow A_s\alpha \in P_4$
faça excluir $A_r \rightarrow A_s\ \alpha$ de P_4;
para toda $A_s \rightarrow \beta$ de P_4
faça $P_4 = P_4 \cup \{ A_r \rightarrow \beta\ \alpha \}$;

- produções relativas às variáveis auxiliares B_r
  - iniciam por um terminal do lado direito

```
para   toda Br → As βr
faça   excluir Br → As βr de P4;
       para   toda As → a α
       faça   P4 = P4 ∪ { Br → a α βr };
```

Etapa 6: produções na forma $A \rightarrow a\alpha$, α palavra de variáveis

- análoga à correspondente etapa da Forma Normal de Chomsky

Exp: Algoritmo: Forma Normal de Greibach

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

- $P = \{ S \rightarrow AA \mid a, A \rightarrow SS \mid b \}$

Etapa 1: simplificação da gramática

- já está simplificada

Etapa 2: renomeação das variáveis em uma ordem crescente qualquer

- S e A são renomeadas para A_1 e A_2
  - $A_1 \rightarrow A_2A_2 \mid a$
  - $A_2 \rightarrow A_1A_1 \mid b$

Etapas 3 e 4: transformação de produções para a forma $A_r \rightarrow A_s\alpha$, na qual $r \leq s$ e *exclusão das recursões da forma $Ar \rightarrow A_r\alpha$*

- $A_2 \rightarrow A_1A_1$ necessita ser modificada, resultando em
  - $A_1 \rightarrow A_2A_2 \mid a$
  - $A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

- $A_2 \rightarrow A_2A_2A_1$ contém recursão (variável auxiliar B)
  - $A_1 \rightarrow A_2A_2 \mid a$
  - $A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB$
  - $B \rightarrow A_2A_1 \mid A_2A_1B$

Etapa 5: um terminal no início do lado direito de cada produção

- lado direito das produções da maior variável A_2
  - inicia por um terminal
- substitui A_2 em $A_1 \rightarrow A_2A_2$ pelas correspondentes derivações
  - $A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1BA_2 \mid bBA_2 \mid a$
  - $A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB$
  - $B \rightarrow A_2A_1 \mid A_2A_1B$
- produções referentes à variável B

$$B \rightarrow aA_1A_1 \mid bA_1 \mid aA_1BA_1 \mid bBA_1 \mid$$
$$aA_1A_1B \mid bA_1B \mid aA_1BA_1B \mid bBA_1B$$

Etapa 6: produções na forma $A \rightarrow a\alpha$, α composta por variáveis

- produções já estão nessa forma

Gramática resultante, na Forma Normal de Greibach

$$G_{FNG} = (\{ A_1, A_2, B \}, \{ a, b \}, P_{FNG}, A_1),$$

- $P_{FNG} = \{$
  - $A_1 \rightarrow aA_1A_2 \mid bA_2 \mid aA_1BA_2 \mid bBA_2 \mid a,$
  - $A_2 \rightarrow aA_1 \mid b \mid aA_1B \mid bB,$
  - $B \rightarrow aA_1A_1 \mid bA_1 \mid aA_1BA_1 \mid bBA_1 \mid aA_1A_1B \mid bA_1B \mid aA_1BA_1B \mid bBA_1B \}$

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha

6.6 Recursão à Esquerda

- **Recursão à esquerda**

$$A \Rightarrow^{+} A\alpha$$

- **Freqüentemente é desejável que a gramática não seja recursiva à esquerda**
  - exemplo: desenvolvimento de algoritmos reconhecedores
- **Algoritmo**
  - quatro primeiras etapas do algoritmo Forma Normal de Greibach

Def: Algoritmo: Gramática sem Recursões à Esquerda

$G = (V, T, P, S)$ GLC

Etapa 1: simplificação da gramática

Etapa 2: renomeação das variáveis em uma ordem crescente qualquer

Etapa 3: produções na forma $A_r \rightarrow A_s\alpha$, *na qual* $r \leq s$

Etapa 4: exclusão das recursões da forma $A_r \rightarrow A_r\alpha$

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha
6.7.1 Definição do Autômato com Pilha
6.7.2 Autômato com Pilha e Linguagens Livres do Contexto
6.7.3 Número de Pilhas e o Poder Computacional

6.7 Autômato com Pilha

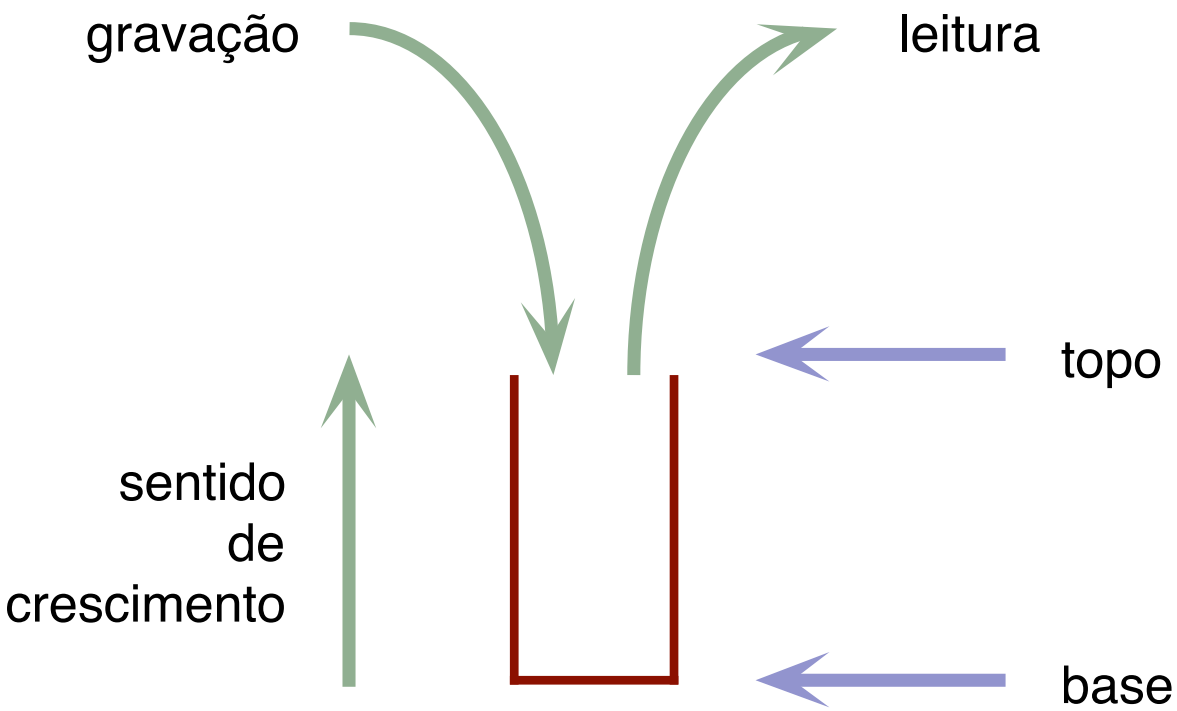
- **Classe das Linguagens Livres do Contexto**
  - pode ser associada a um formalismo do tipo autômato
  - Autômato com Pilha
- **Autômato com pilha**
  - análogo ao autômato finito
  - incluindo uma pilha como memória auxiliar
  - não-determinismo

◆ Pilha

- independente da fita de entrada
- *não* possui limite máximo de tamanho
  - “tão grande quanto se queira”
  - baseada na noção de conjunto *infinitamente contável*

◆ Estrutura de uma pilha

- último símbolo gravado é o primeiro a ser lido
- base: fixa e define o seu início
- topo: variável e define a posição do último símbolo gravado



◆ Não-determinismo: importante e necessário

- *aumenta* o poder computacional dos AP
- exemplo

$$\{ ww^r \mid w \text{ é palavra sobre } \{ a, b \} \}$$

  - reconhecimento só é possível por um AP Não-Determinístico

AP × Número de estados

- qualquer LLC pode ser reconhecida por um AP
  - com somente um estado
  - (ou três estados, dependendo da definição)
- pilha é suficiente como única memória
  - estados não são necessários
  - para "memorizar" informações passadas
- estados no AP
  - poderiam ser excluídos
  - sem se reduzir o poder computacional
- como a pilha não possui tamanho máximo
  - AP pode assumir tantos estados quanto se queira

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha
6.7.1 Definição do Autômato com Pilha
6.7.2 Autômato com Pilha e Linguagens Livres do Contexto
6.7.3 Número de Pilhas e o Poder Computacional

6.7.1 Definição do Autômato com Pilha

- **Duas definições universalmente aceitas**
  - estados finais
    - pára aceitando ao atingir um estado final
    - inicialmente a pilha é vazia
  - pilha vazia
    - pára aceitando quando a pilha estiver vazia
    - inicialmente, a pilha um símbolo inicial da pilha
    - não existem estados finais
  - definições equivalentes (possuem o mesmo poder computacional)
    - *adotada* a definição que usa estados finais

◆ AP Não-Determinístico ou simplesmente AP

- Fita
  - análoga à do autômato finito
- Pilha
  - memória auxiliar
  - pode ser usada para leitura e gravação
- Unidade de Controle
  - reflete o estado corrente da máquina
  - possui: cabeça de fita e cabeça de pilha
- Programa, Função Programa ou Função de Transição comanda
  - leitura da fita
  - leitura e gravação da pilha
  - define o estado da máquina

◆ Pilha

- cada célula armazena um símbolo do alfabeto auxiliar
  - pode ser igual ao alfabeto de entrada
- leitura ou gravação é sempre no topo
- não possui tamanho fixo, nem máximo
  - tamanho corrente: tamanho da palavra armazenada
  - valor inicial: vazio (palavra vazia)

◆ Unidade de controle

- número *finito* e *predefinido* de estados
- Cabeça da Fita
  - unidade de leitura: acessa uma célula da fita de cada vez
  - move exclusivamente para a direita
  - pode testar se a entrada foi completamente lida
- Cabeça da Pilha
  - unidade de leitura e gravação

◆ Cabeça da Pilha: leitura e gravação

- Leitura
  - move para a direita ("para baixo") ao ler um símbolo
  - acessa um símbolo de cada vez, sempre do topo
  - exclui o símbolo lido
  - pode testar se a pilha está vazia

- Gravação
  - move para a esquerda ("para cima") ao gravar
  - pode gravar uma palavra composta por mais de um símbolo
  - símbolo do topo é o mais à esquerda da palavra gravada

◆ Controle Finito?

- Unidade de controle: número *finito* e *predefinido* de estados
- Mas não é dita de controle finito
  - (em oposição aos autômatos finitos)
  - conteúdo da pilha também caracteriza o estado do sistema

◆ Programa é uma função parcial

- dependendo
  - estado corrente
  - símbolo lido da fita
  - símbolo lido da pilha
- determina
  - novo estado
  - palavra a ser gravada (na pilha)
- possui a facilidade de movimento vazio
  - permite mudar de estado sem ler da fita

Def: Autômato com Pilha (Não-Determinístico)

$$M = (\Sigma, Q, \delta, q_0, F, V)$$

- Σ - alfabeto de símbolos) de entrada
- Q - conjunto de estados possíveis o qual é finito
- δ - (função) programa ou função de transição
  - função parcial

$$\delta: Q \times (\Sigma \cup \{ \varepsilon, ? \}) \times (V \cup \{ \varepsilon, ? \}) \rightarrow 2^{Q \times V^*}$$

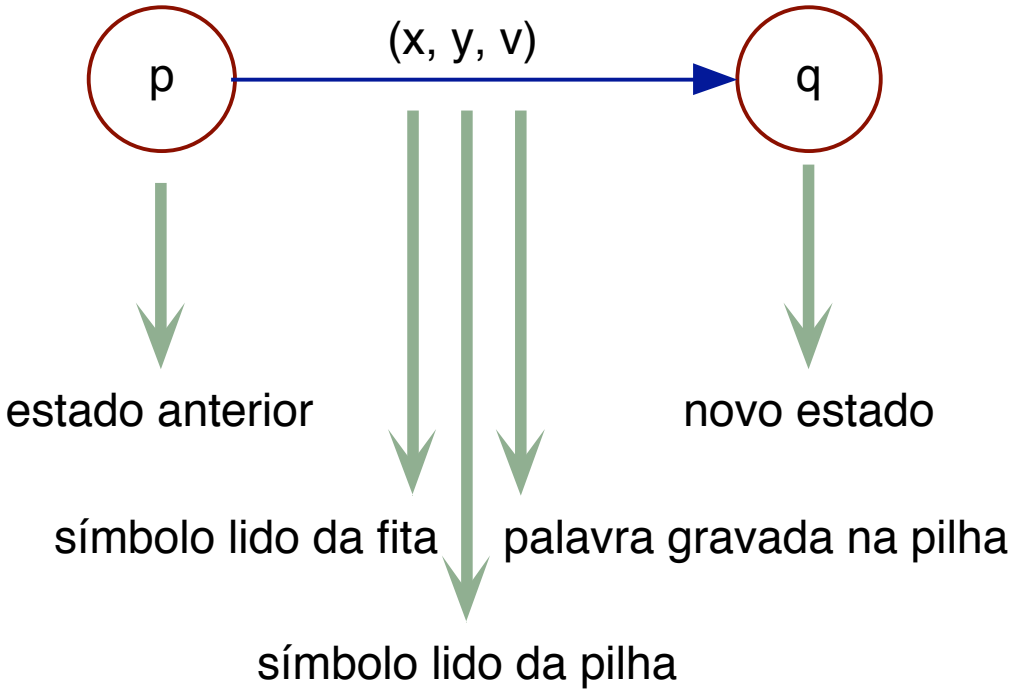
$$\delta(p, x, y) = \{ (q_1, v_1), \ldots, (q_n, v_n) \}$$ transição

- q_0 - elemento distinguido de Q: estado inicial
- F - subconjunto de Q: conjunto de estados finais
- V - alfabeto auxiliar ou alfabeto da pilha

◆ Características da função programa

- função parcial
- "?" indica teste de
  - pilha vazia
  - toda palavra de entrada lida
- leitura de ε indica
  - movimento vazio da fita ou pilha (não lê, nem move a cabeça)
  - não-determinístico: basta que o movimento seja vazio na fita
- gravação de ε
  - nenhuma gravação é realizada na pilha (e não move a cabeça)

- **Exemplo:** $\delta(p, ?, \varepsilon) = \{ (q, \varepsilon) \}$
  - no estado p, se a entrada foi completamente lida, não lê da pilha
  - assume o estado q e não grava na pilha
- **Programa como diagrama:** $\delta(p, x, y) = \{ (q, v) \}$



◆ Computação de um AP

- sucessiva aplicação da função programa
  - para cada símbolo da entrada (da esquerda para a direita)
  - até ocorrer uma condição de parada

- é possível que *nunca* atinja uma condição de parada
  - processa indefinidamente (*loop* infinito)
  - exemplo: empilha e desempilha um mesmo símbolo indefinidamente, sem ler da fita

- definição formal
  - estende a definição da função programa
  - argumento: um estado e uma palavra
  - exercício

◆ Parada de um AP

- Aceita
  - *pelo menos um* dos caminhos alternativos atinge um estado final
  - não importa se leu ou não toda a entrada
- Rejeita
  - *todos* os caminhos alternativos rejeitam a entrada
  - a função programa é indefinida para cada caso
- Loop
  - *pelo menos um* caminho alternativo está em *loop* infinito
  - demais: rejeitam ou também estão em *loop* infinito

Def: Linguagem Aceita, Rejeitada, Loop

$M = (\Sigma, Q, \delta, q_0, F, V)$ autômato com pilha

Linguagem Aceita ou Linguagem Reconhecida: ACEITA(M) ou L(M)

- todas as palavras de Σ^* aceitas por M, a partir de q_0

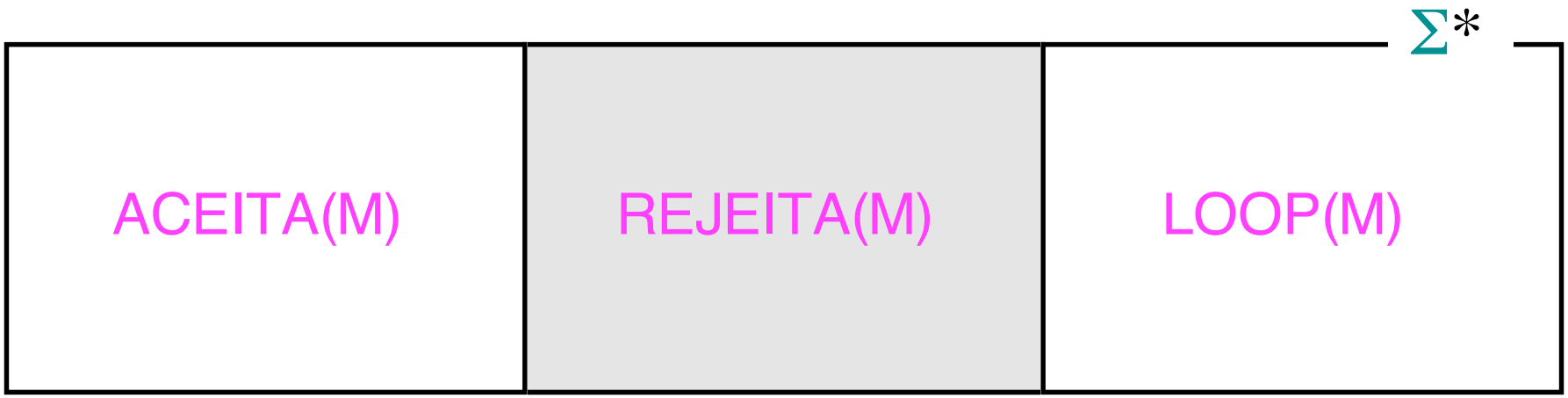
Linguagem Rejeitada: REJEITA(M)

- todas as palavras de Σ^* rejeitadas por M, a partir de q_0

Linguagem Loop: LOOP(M)

- todas as palavras de Σ^* para as quais M fica processando indefinidamente a partir de q_0

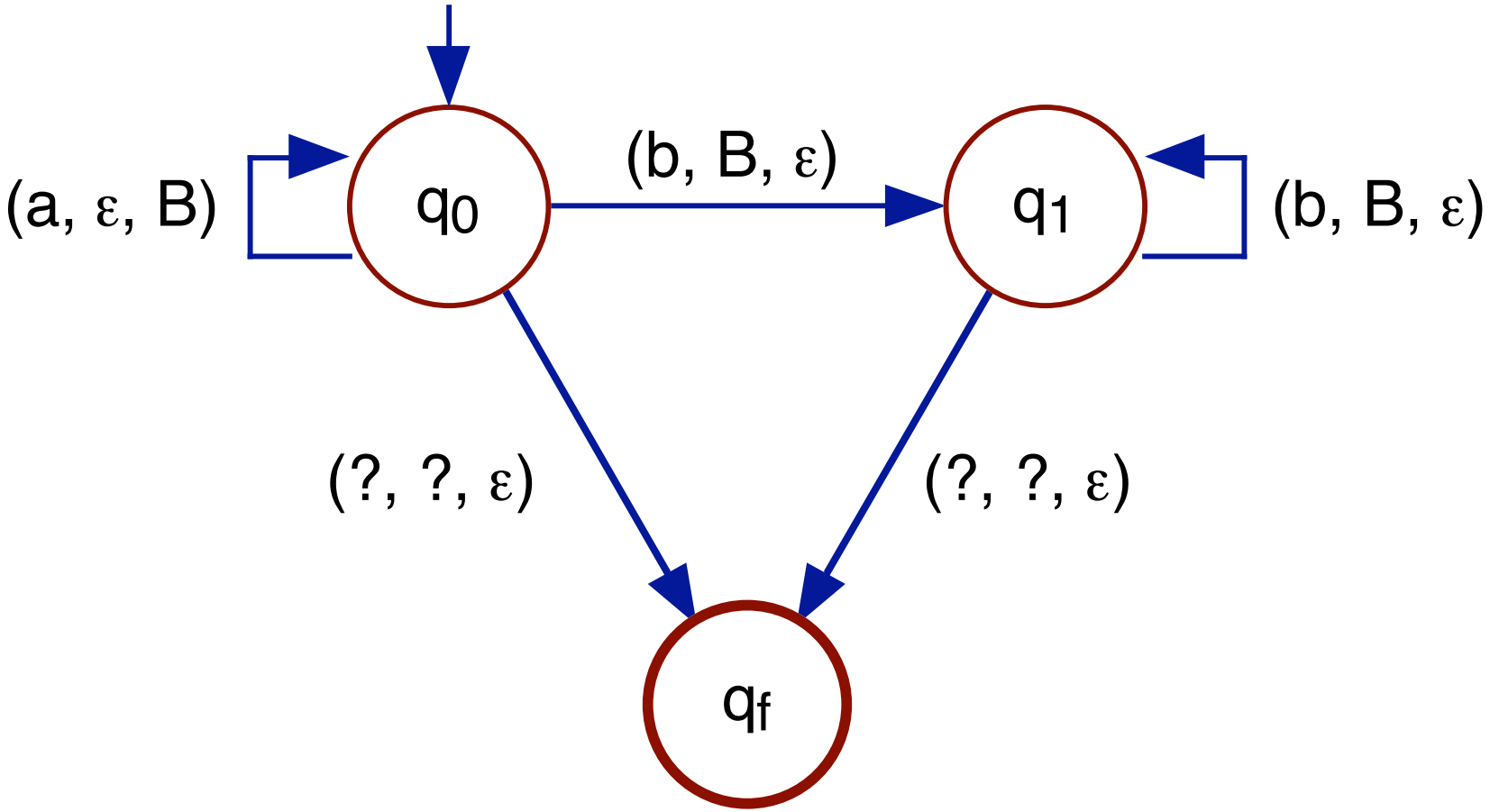
◆ Partição de Σ^* induzida por um AP M



- algum conjunto vazio?
  - partição induzida contém um conjuntos a menos
  - uma classe de equivalência não pode ser vazia

Exp: Autômato com Pilha: Duplo Balanceamento

$$M_1 = (\{ a, b \}, \{ q_0, q_1, q_f \}, \delta_1, q_0, \{ q_f \}, \{ B \})$$



Exp: Autômato com Pilha: Duplo Balanceamento

$$M_1 = (\{ a, b \}, \{ q_0, q_1, q_f \}, \delta_1, q_0, \{ q_f \}, \{ B \})$$

AP determinístico

$ACEITA(M_1) = L_1$

$LOOP(M_1)$ é vazio?

- $\delta_1 (q_0, a, \varepsilon) = \{ (q_0, B) \}$
- $\delta_1 (q_0, b, B) = \{ (q_1, \varepsilon) \}$
- $\delta_1 (q_0, ?, ?) = \{ (q_f, \varepsilon) \}$
- $\delta_1 (q_1, b, B) = \{ (q_1, \varepsilon) \}$
- $\delta_1 (q_1, ?, ?) = \{ (q_f, \varepsilon) \}$

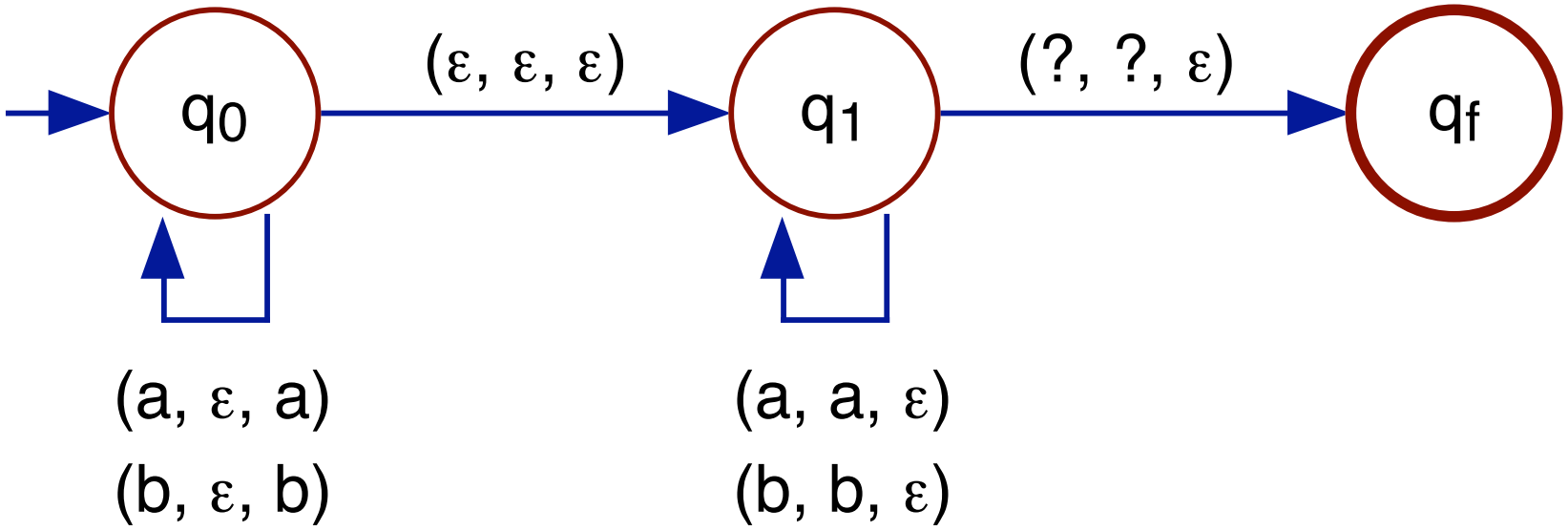
Exp: Autômato com Pilha: Palavra e sua Reversa

$$L_3 = \{ ww^r \mid w \text{ pertence a } \{ a, b \}^* \}$$

$ACEITA(M_3) = L_3$

$LOOP(M_3)$ é vazio?

AP não-determinístico (por quê?)



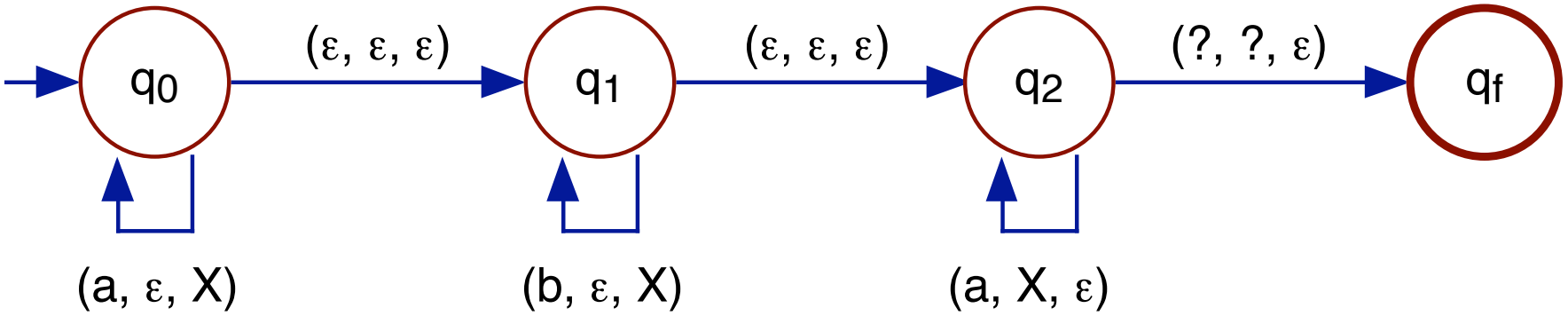
Exp: Autômato com Pilha: $a^n b^m a^{n+m}$

$$L_4 = \{ a^n b^m a^{n+m} \mid n \geq 0, m \geq 0 \}$$

$ACEITA(M_4) = L_4$

$LOOP(M_4)$ é vazio?

AP não-determinístico (por quê?)



6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha
6.7.1 Definição do Autômato com Pilha
6.7.2 Autômato com Pilha e Linguagens Livres do Contexto
6.7.3 Número de Pilhas e o Poder Computacional

6.7.2 AP e Linguagens Livres do Contexto

- **Classe linguagens aceitas por AP = Classe LLC**
  - classe das linguagens geradas pelas GLC
- **Construção de um AP a partir de uma GLC qualquer, permite concluir**
  - construção de um reconhecedor para uma LLC a partir de sua gramática é simples e imediata
  - qualquer LLC pode ser aceita por um AP com somente um estado
    - estados não aumentam o poder computacional

Teorema: GLC → AP

Se L é uma LLC, então existe M, AP M tal que ACEITA(M) = L

Prova: Suponha que $\varepsilon \notin L$

Construção de um AP a partir da gramática na FNG

- produções da forma $A \rightarrow a\alpha$, α palavra de variáveis

AP resultante simula a derivação mais à esquerda

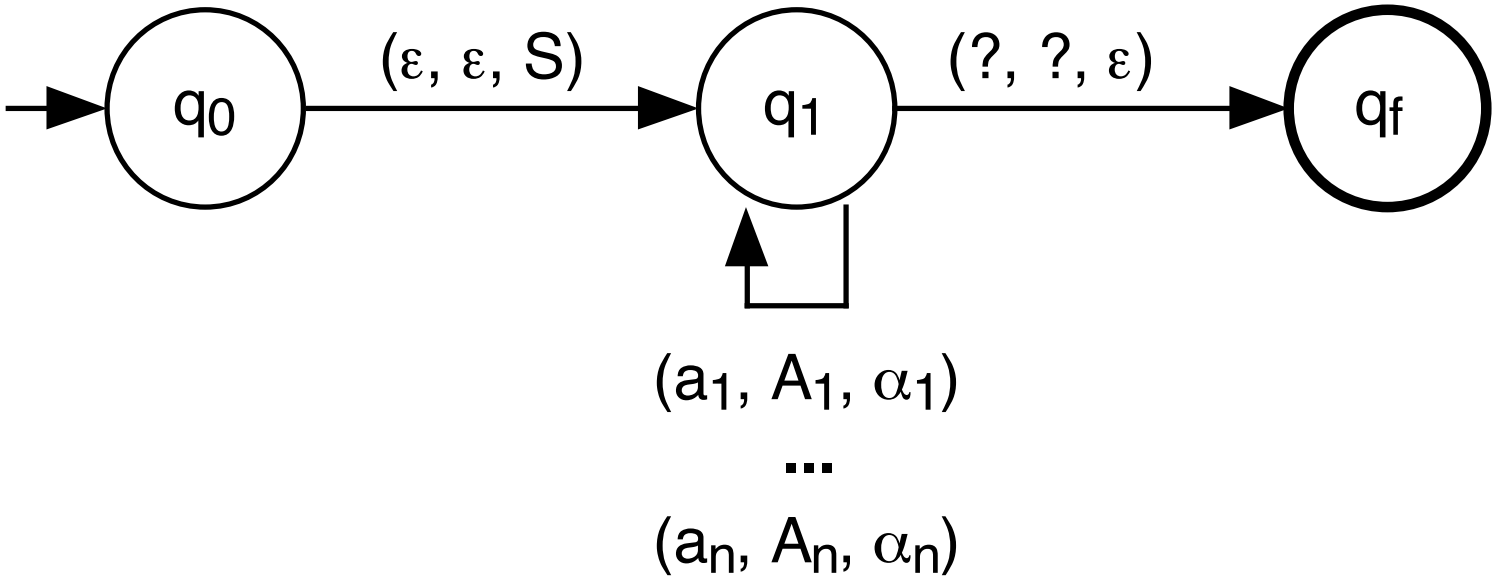
- lê o símbolo a da fita
- lê o símbolo A da pilha
- empilha a palavra de variáveis α

AP M a partir da gramática G = (V, T, P, S)

$G_{FNG} = (V_{FNG}, T_{FNG}, P_{FNG}, S)$, é G na Forma Normal de Greibach

$$M = (T_{FNG}, \{ q_0, q_1, q_f \}, \delta, q_0, \{ q_f \}, V_{FNG})$$

- $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$
- $\delta(q_1, a, A) = \{ (q_1, \alpha) \mid A \rightarrow a\alpha \in P_{FNG} \}$
- $\delta(q_1, ?, ?) = \{ (q_f, \varepsilon) \}$



A demonstração de que $ACEITA(M) = GERA(G_{FNG})$

- indução no número de movimentos de M (ou derivações de G_{FNG})
  - exercício
- como o AP pode ser modificado para tratar a palavra vazia?

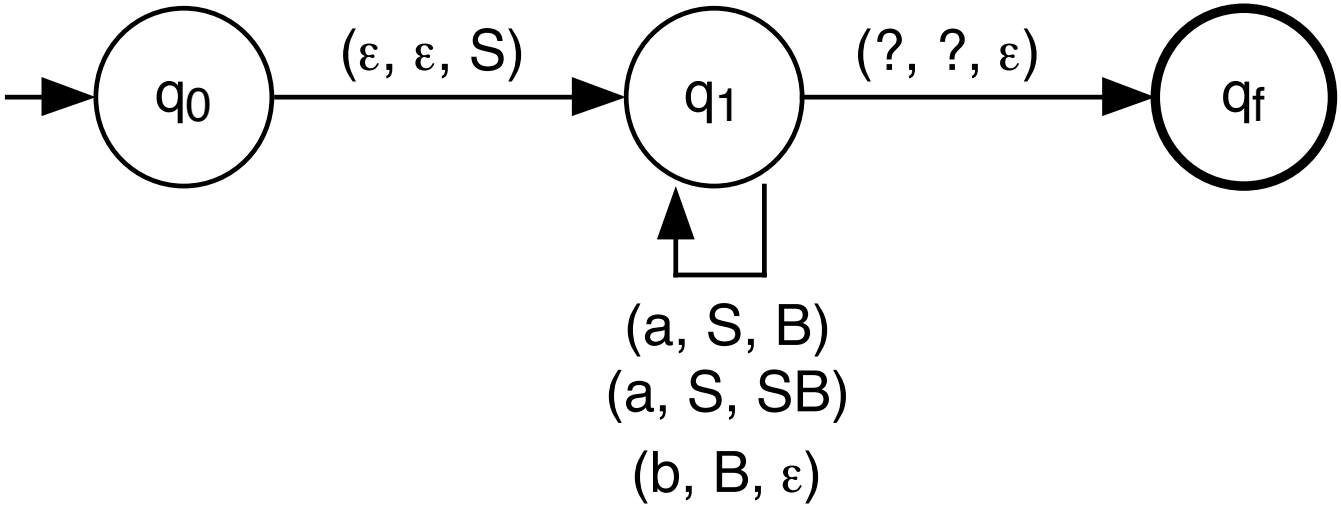
Exp: GLC → AP: $L_5 = \{ a^n b^n \mid n \geq 1 \}$

Gramática na Forma Normal de Greibach

- $G_5 = (\{ S, B \}, \{ a, b \}, P_5, S)$
- $P_5 = \{ S \rightarrow aB \mid aSB, B \rightarrow b \}$

Correspondete AP

$$M_5 = (\{ a, b \}, \{ q_0, q, q_f \}, \delta_5, q_0, \{ q_f \}, \{ S, B \})$$



Corolário : AP × Número de Estados

Se L é uma LLC, então existe M

- AP com controle de aceitação por estados finais, com três estados, tal que ACEITA(M) = L
- AP com controle de aceitação por pilha vazia, com um estado tal que ACEITA(M) = L

Corolário : Existência de um AP que Sempre Pára

Se L é uma LLC, então existe M, AP, tal que

- ACEITA(M) = L
- REJEITA(M) = Σ^* - L
- LOOP(M) = $\varnothing$

Teorema: AP → GLC

Se L é aceita por um AP, então L é LLC

- demonstração omitida

Obs: Estados × Poder Computacional dos AP

A combinação dos resultados:

- Corolário: AP × Número de Estados
- Corolário: Existência de um AP que Sempre Pára
- Teorema: AP → GLC

comprovam que o uso dos estados como "memória" não aumenta o poder de reconhecimento do AP

6 – Linguagens Livres do Contexto

6.1 Gramática Livre do Contexto
6.2 Árvore de Derivação
6.3 Gramática Livre do Contexto Ambígua
6.4 Simplificação de Gramática Livre do Contexto
6.5 Formas Normais
6.6 Recursão à Esquerda
6.7 Autômato com Pilha
6.7.1 Definição do Autômato com Pilha
6.7.2 Autômato com Pilha e Linguagens Livres do Contexto
6.7.3 Número de Pilhas e o Poder Computacional

6.7.3 Número de Pilhas e o Poder Computacional

◆ Modelo autômato com pilha

- Adequado para estudos aplicados e formais
  - pilha é adequada para implementação em computadores
  - poucas modificações na definição determinam significativas alterações no poder computacional

- principais estudos de linguagens e computabilidade
  - podem ser desenvolvidos usando-se exclusivamente AP
  - variando o número de pilhas
  - com ou sem não-determinismo

Autômato com Pilha, sem usar a estrutura de pilha

- estados: única forma de memorizar informações passadas
- muito semelhante ao autômato finito
- AP, sem usar a pilha, com ou sem não-determinismo
  - reconhecem a Classe das Linguagens Regulares

Autômato com Pilha Determinístico

- aceita a Classe das Linguagens Livres do Contexto Determinísticas
  - importante subconjunto próprio da Classe das LLC
- implementação de um AP determinístico é simples e *eficiente*
  - facilita o desenvolvimento de analisadores sintáticos
- algumas propriedades da Classe das LLC Determinísticas
  - existe um tipo de gramática que gera exatamente tal classe (exercício de pesquisa)
  - é fechada para a operação de complemento
  - *não* é fechada para as operações de união, intersecção e concatenação

Autômato com (uma) Pilha Não-Determinístico

- aceitam exatamente a Classe das LLC

Autômato com Duas Pilhas

- mesmo poder computacional da Máquina de Turing
  - considerada o dispositivo mais geral de computação
- se existe um algoritmo para resolver um problema
  - pode ser expresso como um autômato com duas pilhas
- não-determinismo não aumenta o poder computacional

Autômato com Múltiplas Pilhas

- poder computacional de um autômato com mais de duas pilhas
  - equivalente ao do autômato com duas pilhas
- se um problema é solucionado por um autômato com múltiplas pilhas
  - pode ser solucionado por um autômato com duas pilhas

Linguagens Formais e Autômatos

P. Blauth Menezes

1 Introdução e Conceitos Básicos
2 Linguagens e Gramáticas
3 Linguagens Regulares
4 Propriedades das Linguagens Regulares
5 Autômato Finito com Saída
6 Linguagens Livres do Contexto
7 Propriedades e Reconhecimento das Linguagens Livres do Contexto
8 Linguagens Recursivamente Enumeráveis e Sensíveis ao Contexto
9 Hierarquia de Classes e Linguagens e Conclusões

Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

Departamento de Informática Teórica

Instituto de Informática / UFRGS