Teoria da Computação

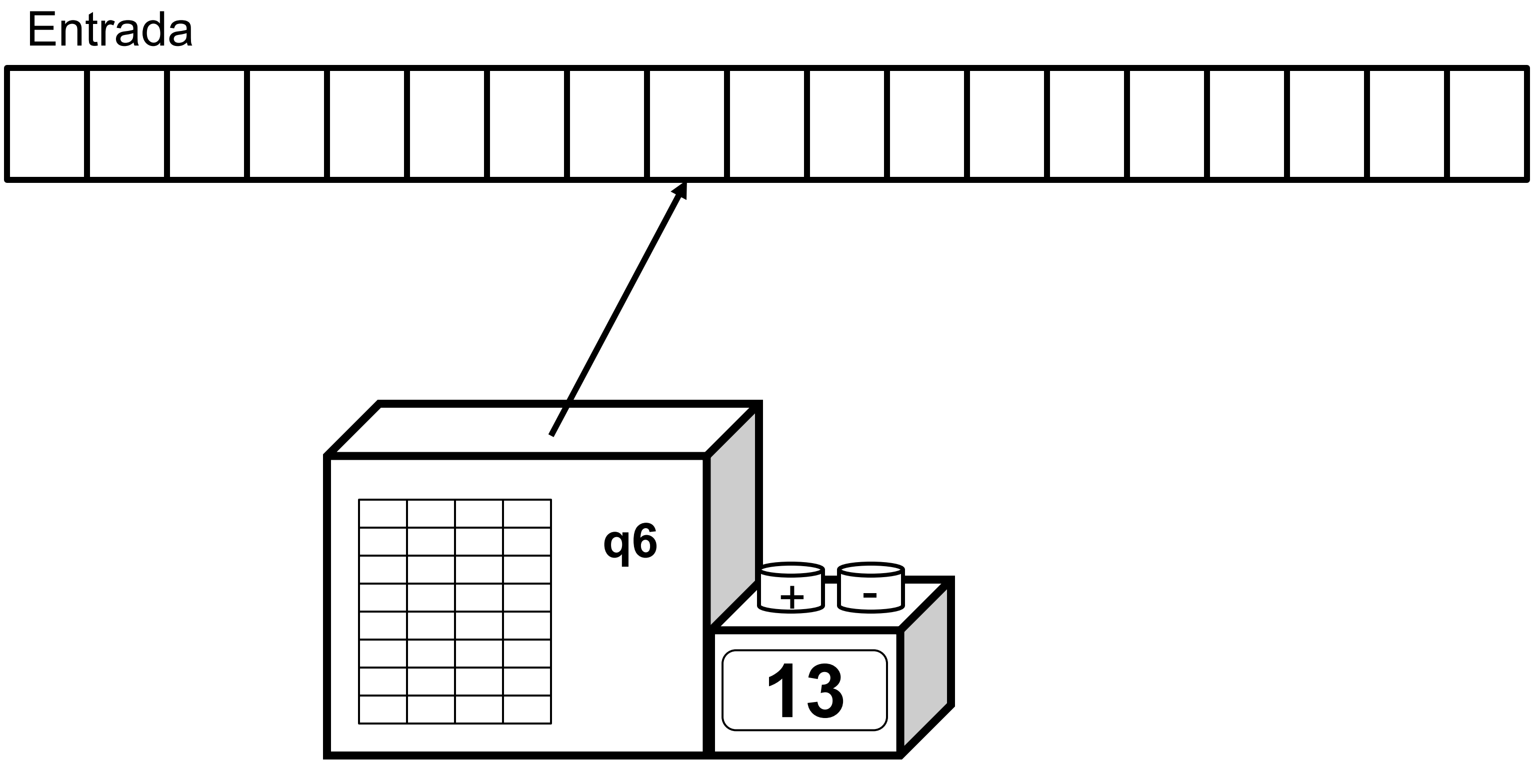
Prof. Sergio D Zorzo
Departamento de Computação - UFSCar
4

Linguagens livres de contexto

Linguagens livres de contexto

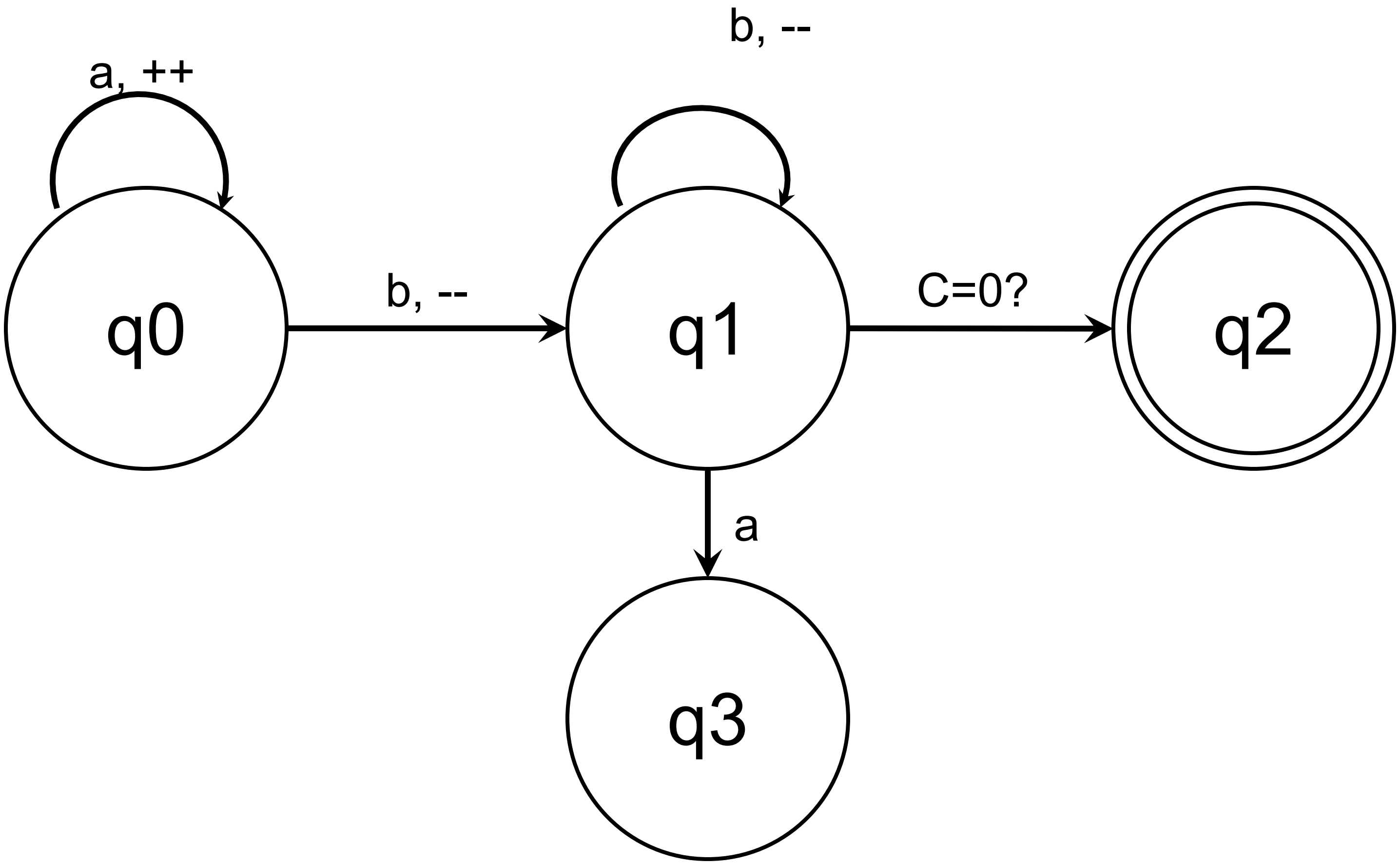
- Linguagens regulares permitem descrever muitas coisas práticas
  - Ex: busca textual, mecanismos simples de comunicação, máquinas e protocolos simples
- Mas são limitadas
  - “Não conseguem contar” → autômatos finitos
- Mas e se adicionarmos um contador aos autômatos finitos?

Linguagens livres de contexto



Linguagens livres de contexto

- Ex: $a^n b^n$



Linguagens livres de contexto

- Uma classe maior de linguagens
  - Linguagens livres de contexto

- Inicialmente, foram uma maneira de entender a linguagem humana
  - Gramáticas livres de contexto
    - Formalização das regras gramaticais da linguagem humana

- Característica principal: recursão
  - Ex: linguagens naturais: frases nominais dentro de frases verbais, e vice-versa

Linguagens livres de contexto

- A partir do estudo da linguagem humana, chegou-se a um modelo matemático formal sobre uma classe de linguagens
- O conceito é o mesmo das linguagens regulares
  - Linguagens = problemas
  - Um problema é:
    - Dada uma cadeia, determinar se pertence ou não à linguagem
    - A diferença é que aqui, o conceito cadeia é diferente ...
    - ... e as regras de definição da linguagem são mais poderosas do que simples transições em um autômato finito

Linguagens livres de contexto

- Podemos comparar linguagens regulares e linguagens livres de contexto nos seguintes aspectos:
  - Linguagens regulares:
    - Base: símbolos de um alfabeto
    - Regras (“gramática”): expressões regulares
    - Característica:
      - estados finitos / não conseguem contar / lema do bombeamento para linguagens regulares
  - Linguagens livres de contexto:
    - Base: símbolos terminais
      - Pode-se pensar que um símbolo terminal é um símbolo de um alfabeto (Mas cada símbolo terminal pode ser uma cadeia sobre uma linguagem regular - uma palavra)
    - Regras (“gramática”): gramáticas livres de contexto
    - Característica:
      - recursão simples / consegue contar (de forma limitada) / lema do bombeamento para linguagens livres de contexto

Gramáticas livres de contexto

Gramáticas livres de contexto

- As linguagens livres de contexto são descritas por gramáticas livres de contexto
- As regras de produção tem leis de formação específica

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Regras de substituição
ou
produções

Gramáticas livres de contexto

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

Lado esquerdo ou **cabeça**: sempre um único símbolo. Esses símbolos são chamados de **variáveis** ou **não-terminais**

Lado direito ou **corpo**: uma cadeia de símbolos. Podem ter variáveis e outros símbolos, chamados de **terminais**

Uma das variáveis é designada como a variável ou símbolo inicial. É a variável que aparece do lado esquerdo da primeira regra. (Neste exemplo, **A** é o símbolo inicial)

Gramáticas livres de contexto

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

$$A \rightarrow 0A1 \mid B$$
$$B \rightarrow \#$$

Sempre que houver mais de uma produção para uma mesma variável, podemos agrupá-las com o símbolo “|”.

Gramáticas livres de contexto

A GLC G é definida pela quadrupla (V,T,P,S)

- V = conjunto de variáveis
- T = conjunto de terminais
- P = conjunto de produções
- S = símbolo inicial

Ex: $G_{palíndromos} = (\{P\},\{0,1\},A,P)$

- A = {
  - $P \rightarrow \varepsilon$
  - $P \rightarrow 0$
  - $P \rightarrow 1$
  - $P \rightarrow 0P0$
  - $P \rightarrow 1P1$
- }

Gramáticas livres de contexto

- Como uma gramática descreve uma linguagem?
- Duas formas:
  - Inferência recursiva
  - Derivação
- Ex: Gramática para expressões aritméticas
  - V = {E,I}
  - T = {+,*,(,),a,b,0,1}
  - P = conjunto de regras ao lado
  - S = E

$$
\begin{aligned}
E &\rightarrow I \\
E &\rightarrow E + E \\
E &\rightarrow E * E \\
E &\rightarrow (E) \\
I &\rightarrow a \\
I &\rightarrow b \\
I &\rightarrow Ia \\
I &\rightarrow Ib \\
I &\rightarrow I0 \\
I &\rightarrow I1
\end{aligned}
$$

Gramáticas livres de contexto

- Inferência recursiva
  - Dada uma cadeia (conjunto de símbolos terminais)
  - Do corpo para a cabeça
- Ex: a*(a+b00)
  - a*(a+b00) ⇐ a*(a+I00) ⇐ a*(a+I0) ⇐ a*(a+I) ⇐ a*(a+E) ⇐ a*(I+E) ⇐ a*(E+E) ⇐ a*(E) ⇐ a*E ⇐ I*E ⇐ E*E ⇐ E

Gramáticas livres de contexto

- Derivação
  - Dada uma cadeia (conjunto de símbolos terminais)
  - Da cabeça para o corpo
- Ex: a*(a+b00)
  - $E \Rightarrow E*E \Rightarrow I*E \Rightarrow a*E \Rightarrow a*(E) \Rightarrow a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(a+E) \Rightarrow a*(a+I) \Rightarrow a*(a+I0) \Rightarrow a*(a+I00) \Rightarrow a*(a+b00)$
- Símbolo de derivação: $\Rightarrow$
- Derivação em múltiplas etapas: $\Rightarrow^*$ (obs: asterisco acima da seta)
  - $E \Rightarrow^* a*(E)$
  - $a*(E+E) \Rightarrow^* a*(a+I00)$
  - $E \Rightarrow^* a*(a+b00)$

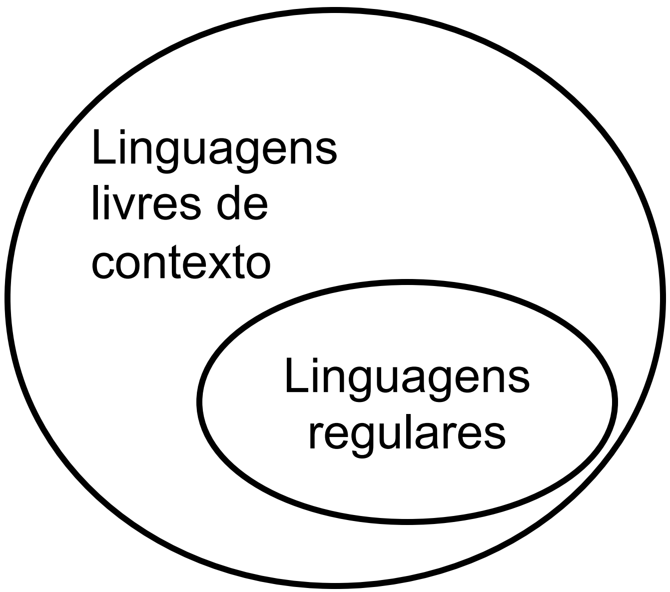
Gramáticas livres de contexto

- Derivações mais à esquerda
  - Sempre substituir a variável mais à esquerda
  - Notação: $\Rightarrow_{lm}$, $\Rightarrow^{*}_{lm}$
- Derivações mais à direita
  - Sempre substituir a variável mais à direita
  - Notação: $\Rightarrow_{rm}$, $\Rightarrow^{*}_{rm}$

Gramáticas livres de contexto

- A linguagem de uma gramática
  - $L(G) = \{w \text{ em } T^* \mid S \Rightarrow^* w\}$
- Se uma linguagem L é L(G) de alguma gramática G livre de contexto
  - L é uma linguagem livre de contexto
    - (CFL – *Context-Free Language*)
- Ex: o conjunto de palíndromos pode ser descrito por uma gramática livre de contexto
  - Portanto o conjunto de palíndromos é uma linguagem livre de contexto

Gramáticas livres de contexto



A classe de linguagens livres de contexto engloba a classe de linguagens regulares

Ou seja: toda linguagem regular é livre de contexto

Formas sentenciais

- Derivações a partir do símbolo inicial
  - Formas sentenciais à esquerda
    - Obtidas somente com derivações mais à esquerda
  - Formas sentenciais à direita
    - Obtidas somente com derivações mais à direita
- Ex: $E \Rightarrow_{lm} E*E \Rightarrow_{lm} I*E \Rightarrow_{lm} a*E \Rightarrow_{lm} a*(E)$
- Ex: $E \Rightarrow_{rm} E*E \Rightarrow_{rm} E*(E) \Rightarrow_{rm} a*(E+E) \Rightarrow_{rm} a*(E+I)$

Exercícios

- Dada a gramática descrita pelas produções a seguir:
  - $S \rightarrow A1B$
  - $A \rightarrow 0A \mid \varepsilon$
  - $B \rightarrow 0B \mid 1B \mid \varepsilon$
- Forneça derivações mais à esquerda e mais à direita das seguintes cadeias:
  - a) 00101
  - b) 1001
  - c) 00011

Exercícios

- Dada a gramática descrita pelas produções ao lado:
- Forneça derivações mais à esquerda e mais à direita das seguintes cadeias:
  - a) a+b*(0+1)
  - b) a+a+b+1
  - c) a+b*a

$$
\begin{aligned}
E &\rightarrow I \\
E &\rightarrow E + E \\
E &\rightarrow E * E \\
E &\rightarrow (E) \\
I &\rightarrow a \\
I &\rightarrow b \\
I &\rightarrow 0 \\
I &\rightarrow 1
\end{aligned}
$$

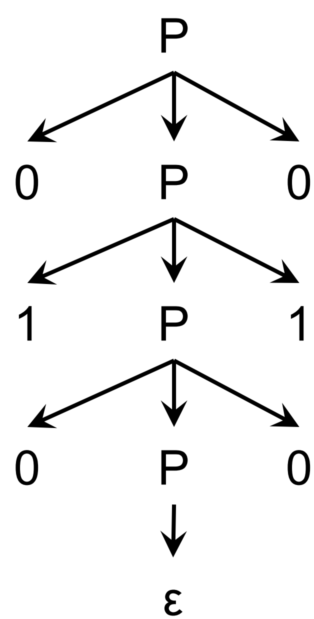
Árvores de análise sintática

Árvores de análise sintática

- Representação visual para derivações
  - Em formato de árvore
- Mostra claramente como os símbolos de uma cadeia de terminais estão agrupados em subcadeias
- Permitem analisar alguns aspectos da linguagem e ver o processo de derivação / inferência recursiva

Árvores de análise sintática

- Ex: palíndromos, cadeia 010010
- Derivações: P ⇒ 0P0 ⇒ 01P10 ⇒010P010 ⇒ 010ε010 = 010010



Árvores de análise sintática

- Seja uma gramática $G = (V,T,P,S)$
- A árvore é construída da seguinte forma:
  - Cada nó interior é rotulado por uma variável em V
  - Cada folha é rotulada por uma variável, um terminal, ou ε. No entanto, se a folha for rotulada por ε, ela deve ser o único filho de seu pai
  - Se um nó interior é rotulado por A e seus filhos são rotulados por

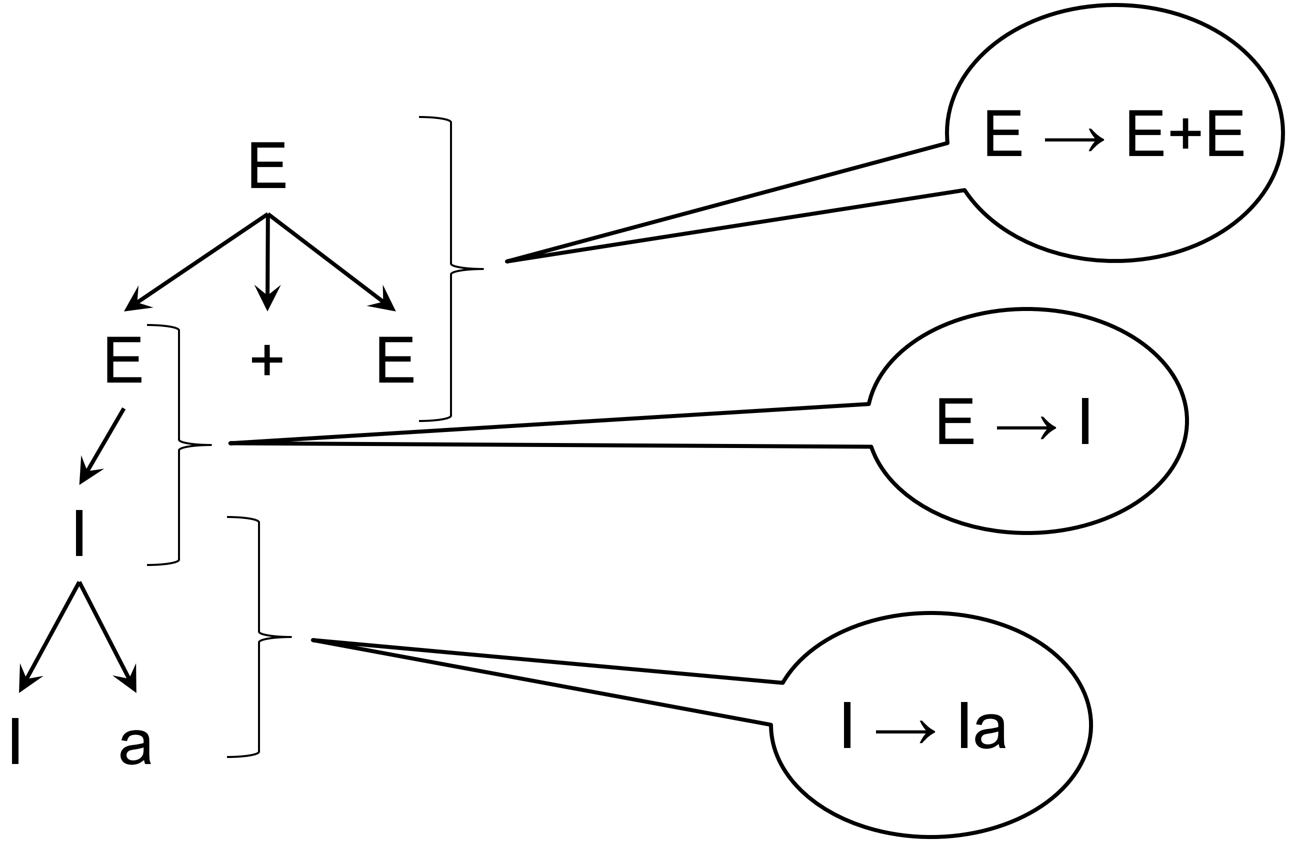
    $$X_1,X_2,\ldots,X_k$$

    Respectivamente, a partir da esquerda, então

    $A \rightarrow X_1X_2\ldots X_k$ é uma produção em P

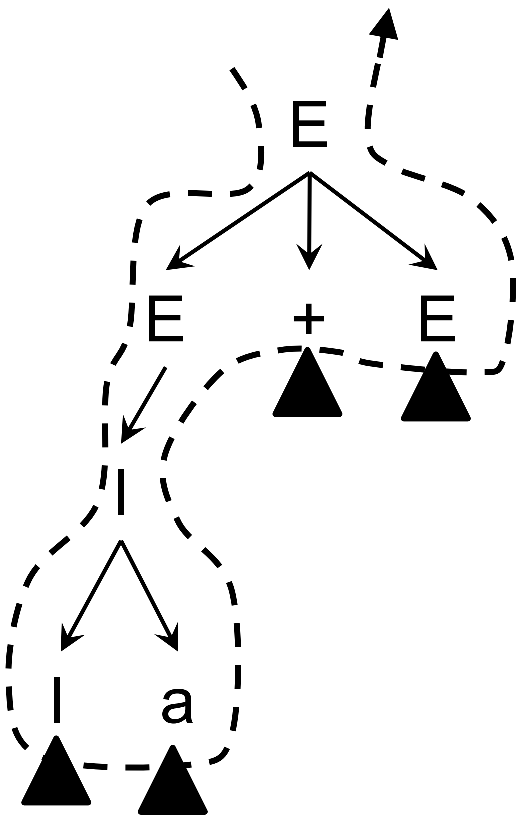
Árvores de análise sintática

- Ex:



O resultado de uma árvore de análise sintática

- Análise das folhas de uma árvore de análise sintática
  - Concatenação dos símbolos a partir da esquerda
    - ***Resultado*** da árvore
    - É sempre uma cadeia derivada da variável raiz



Ex: Ia+E é o resultado da árvore à direita
É também uma derivação a partir da raiz (E)
$E \Rightarrow E+E \Rightarrow I+E \Rightarrow Ia+E$

Árvores de análise sintática

- Árvores especiais
  - O resultado é uma cadeia composta exclusivamente por terminais
    - Isto é, as folhas são rotuladas por um terminal ou ε
  - A raiz é rotulada pelo símbolo inicial
- São árvores cujo resultado é uma cadeia na linguagem da gramática subjacente
  - Este tipo de árvore é útil para analisar alguns aspectos da linguagem, como ambiguidade (veremos mais adiante)

Exercícios

- Dada a gramática descrita pelas produções a seguir:
  - $S \rightarrow A1B$
  - $A \rightarrow 0A \mid \varepsilon$
  - $B \rightarrow 0B \mid 1B \mid \varepsilon$
- Forneça árvores de análise sintática para as seguintes cadeias:
  - a) 00101
  - b) 1001
  - c) 00011

Exercícios

- Dada a gramática descrita pelas produções ao lado:
- Forneça árvores de análise sintática para as seguintes cadeias:
  - a) a+b*(0+1)
  - b) a+a+b+1
  - c) a+b*a

$$
\begin{aligned}
E &\rightarrow I \\
E &\rightarrow E + E \\
E &\rightarrow E * E \\
E &\rightarrow (E) \\
I &\rightarrow a \\
I &\rightarrow b \\
I &\rightarrow 0 \\
I &\rightarrow 1
\end{aligned}
$$

Aplicações das gramáticas livres de contexto

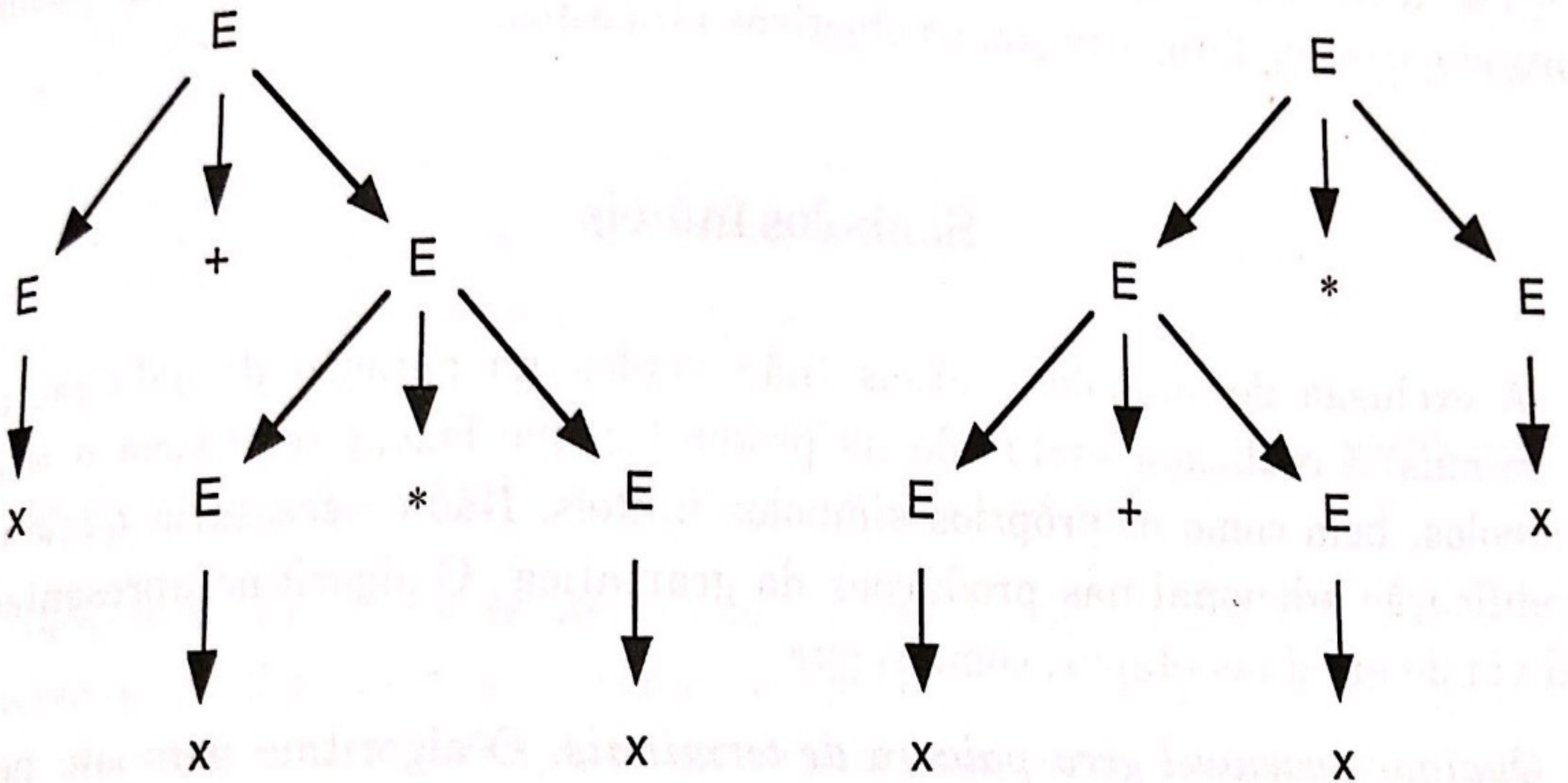
- Foram concebidas originalmente para descrever linguagens naturais
  - Essa promessa não se concretizou
- Mas existem hoje aplicações
  - Descrevem linguagens de programação
    - Existe um modo mecânico de transformar a descrição da linguagem na forma de uma gramática livre de contexto em um analisador sintático, o componente do compilador que descobre a estrutura do código-fonte
    - É uma das primeiras formas de uso das idéias teóricas da ciência da computação

Ambiguidade

- Eventualmente, uma mesma palavra pode ser associada a duas ou mais árvores de derivação, determinando uma ambiguidade. Entretanto nem sempre é possível eliminar ambiguidades. Na realidade, é fácil definir linguagens para as quais qualquer Gramática Livre de Contexto é ambígua, sendo fácil de identificar utilizando derivação mais a esquerda e mais a direita

- E => E+E => x + E=> x+E*E => x+x*E => x+x*x (DME)
- E => E*E => E*x => E+E*x => E+x*x => x+x*x(DMD)

Ambiguidade: Árvores diferentes para uma mesma palavra



Como eliminar Ambiguidade ?

- Não há um processo efetivo e nem que toda gramática ambígua vai ter uma gramática não ambígua equivalente (pois a ambiguidade pode ser inerente a própria linguagem).

Estratégias:

- \- utilizar precedência de operadores
- \- fatorar termos

Formas Normais

Formas Normais

- Estabelecem restrições rígidas na definição das produções, sem reduzir o poder de geração das gramáticas GLC (tipo 2)

- Forma normal de Chomsky onde as regras de produção são dadas por:
  - A-> BC ou A-> a

- Forma normal de Greibach onde as regras de produção são dadas por:
  - A-> aα

    α é uma sequência de variáveis

Forma Normal de Chomsky

- Uma Gramática livre de contexto é dita na forma normal de Chomsky (FNC) se todas as suas regras de produção são da forma :
  - A-> BC ou A-> a A,B,C $\in$ N , a $\in \Sigma$
- O algoritmo para transformação de uma GLC em FNC (linguagem gerada não contém a palavra vazia) tem três etapas:
  - Simplificação da gramática (eliminar símbolos inúteis: estéreis e inacessíveis)
  - Transformação do lado direito das produções de comprimento maior ou igual a dois
  - Transformação do lado direito das produções de comprimento maior ou igual a três, em produções com exatamente duas variáveis

Gramática Livre de Contexto para Forma Normal de Chomsky

- Dada uma gramática livre de contexto G =(V,T,P,S), tal que $\varepsilon \notin L(G)$, tem-se os seguintes passos para a transformação:

- Etapa 1: Simplificação das regras de produção
  - Produções vazias ;
  - Produções da forma A-> B;
  - Símbolos inúteis(opcional);
  - $G_1 = (V_1, T_1, P_1, S)$

Gramática Livre de Contexto para Forma Normal de Chomsky

- Etapa 2: Transformação do lado direito das produções de comprimento maior ou igual a dois
  - $G_2 = (V_2, T_1, P_2, S)$

```
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 pela variável Cₐ em A → X₁X₂...Xₙ ∈ P₂;
             P₂ = P₂ ∪ { Cₐ → a };
```

Gramática Livre de Contexto para Forma Normal de Chomsky

- Etapa 3: Transformação do lado direito das produções de comprimento maior ou igual a três, em produções com exatamente duas variáveis
- $G_3 = (V_3, T_1, P_3, S)$

$V_3 = V_2;$

$P_3 = P_2;$

para toda $A \to B_1B_2 \ldots B_n \in P_3$ **tal que** $n \geq 3$

faça $P_3 = P_3 - \{ A \to B_1B_2 \ldots B_n \};$

$\quad V_3 = V_3 \cup \{ D_1, \ldots, D_{n-2} \};$

$\quad P_3 = P_3 \cup \{ A \to B_1D_1,\ D_1 \to B_2D_2,\ \ldots,$

$\quad\quad D_{n-3} \to B_{n-2}D_{n-2},\ D_{n-2} \to B_{n-1}B_n \};$

Exemplo de Gramática Livre de Contexto para FNC

- G_2= ({E}, {+,*,[,],x}, P_2,E), onde :
- P_2= {E-> E+E | E*E | [E] | x}
- Resolução:
  - E-> EC_+E | EC_*E | $C_[EC_]$ | x
  - C_+ -> +
  - C_* -> *
  - $C_[$ -> [
  - $C_]$ ->]

Exemplo de Gramática Livre de Contexto para FNC

$E \rightarrow EC_{+}E \mid EC_{*}E \mid C_{[}EC_{]} \mid x$ ficará como

- $E \rightarrow ED_1 \mid ED_2 \mid C_{[}D_3 \mid x$
- $D_1 \rightarrow C_{+}E$
- $D_2 \rightarrow C_{*}E$
- $D_3 \rightarrow EC_{]}$

A gramática na Forma Normal de Chomsky fica:

- $G_2' = (\{E, C_{+}, C_{*}, C_{[}, C_{]}, D_1, D_2, D_3\}, \{+, *, [,], x\}, P_2', E)$, onde:
- $P_2' = \{E \rightarrow ED_1 \mid ED_2 \mid C_{[}D_3 \mid x,$

  $D_1 \rightarrow C_{+}E,\ D_2 \rightarrow C_{*}E,\ D_3 \rightarrow EC_{]},$

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

Forma Normal de Greibach

- Uma Gramática livre de contexto é dita na forma Normal de Greibach (FNG) se todas as suas regras de produção são da forma :
  - A-> aα
    - Onde A é uma variável, a é um terminal e α é uma sequência de variáveis

- O algoritmo para transformação de uma GLC em FNG, cuja linguagem gerada não possua a palavra vazia, é dividido nas seguintes etapas:

Forma Normal de Greibach

- Simplificação da gramática
- Renomeação das variáveis em uma ordem crescente qualquer (A_1, A_2, A_3, ...)
- Transformação de produções para a forma $A_r \text{ -> } A_s\alpha$, onde $r \leq s$;
- Exclusão das recursões a esquerda da forma $A_r \text{ -> } A_r\alpha$;

- No final teremos produções na forma $A\text{-> } a\alpha$ onde α é uma sequência de variáveis (pode ser vazia) e a é um terminal.

Gramática Livre de Contexto para Forma Normal de Greibach

- Dada uma gramática livre de contexto G =(V,T,P,S), tal que $\varepsilon \notin L(G)$, tem-se os seguintes passos para a transformação:

- Etapa 1: Simplificação das regras de produção
  - Produções vazias ;
  - Produções da forma A-> B;
  - Símbolos inúteis(opcional);
  - $G_1 = (V_1, T_1, P_1, S)$

Gramática Livre de Contexto para Forma Normal de Greibach

- Etapa 2: Renomeação das variáveis em uma ordem crescente qualquer
  - $G_2 = (V_2, T_1, P_2, S)$
  - $V_2 = \{A_1, A_2, \ldots, A_n\}$
- Etapas 3 e 4:Transformação de produções para a forma $A_r \rightarrow A_s\alpha$, onde $r \leq s$ e exclusão das recursões da forma $A_r \rightarrow A_r\alpha$
  - $G_3 = (V_3, T_1, P_3, S)$

Gramática Livre de Contexto para Forma Normal de Greibach

```
P3 = P2
para   r variando de 1 até n
faça
        para   s variando de 1 até r-1                              Etapa 3
        faça   para   toda Ar → Asα ∈ P3
               faça   excluir Ar → Asα de P3;
                      para   toda As → β ∈ P3
                      faça   P3 = P3 ∪ { Ar → βα }

        para   toda Ar → Arα ∈ P3                                   Etapa 4
        faça   excluir Ar → Arα de P3;
               V3 = V3 ∪ { Br };
               P3 = P3 ∪ { Br → α } ∪ { Br → αBr };
        para   toda Ar → φ ∈ P3 tal que φ não inicia por Ar e
               alguma Ar → Arα foi excluída
        faça   P3 = P3 ∪ { Ar → φBr };
```

Gramática Livre de Contexto para Forma Normal de Greibach

- Etapa 5: Um terminal no inicio do lado direito de cada produção
  - $G_4 = (V_4, T_1, P_4, S)$

```
P4 = P3;
para  r variando de n-1 até 1 e toda Ar → Asα ∈ P4
faça  excluir Ar → Asα de P4;
      para  toda As → β de P4
      faça  P4 = P4 ∪ { Ar → βα };
```

Gramática Livre de Contexto para Forma Normal de Greibach

- Também é necessário garantir que as produções relativas às variáveis auxiliares B_r iniciam por um terminal do lado direito

para toda $B_r \rightarrow A_s\beta_r$
faça excluir $B_r \rightarrow A_s\beta_r$ de P_4;
      para toda $A_s \rightarrow a\alpha$
      faça $P_4 = P_4 \cup \{ B_r \rightarrow a\alpha\beta_r \}$;

- Etapa 6: Produções na forma A-> aα onde α é composta por variáveis. É análoga à correspondente etapa do algoritmo relativo à Forma Normal de Chomsky.

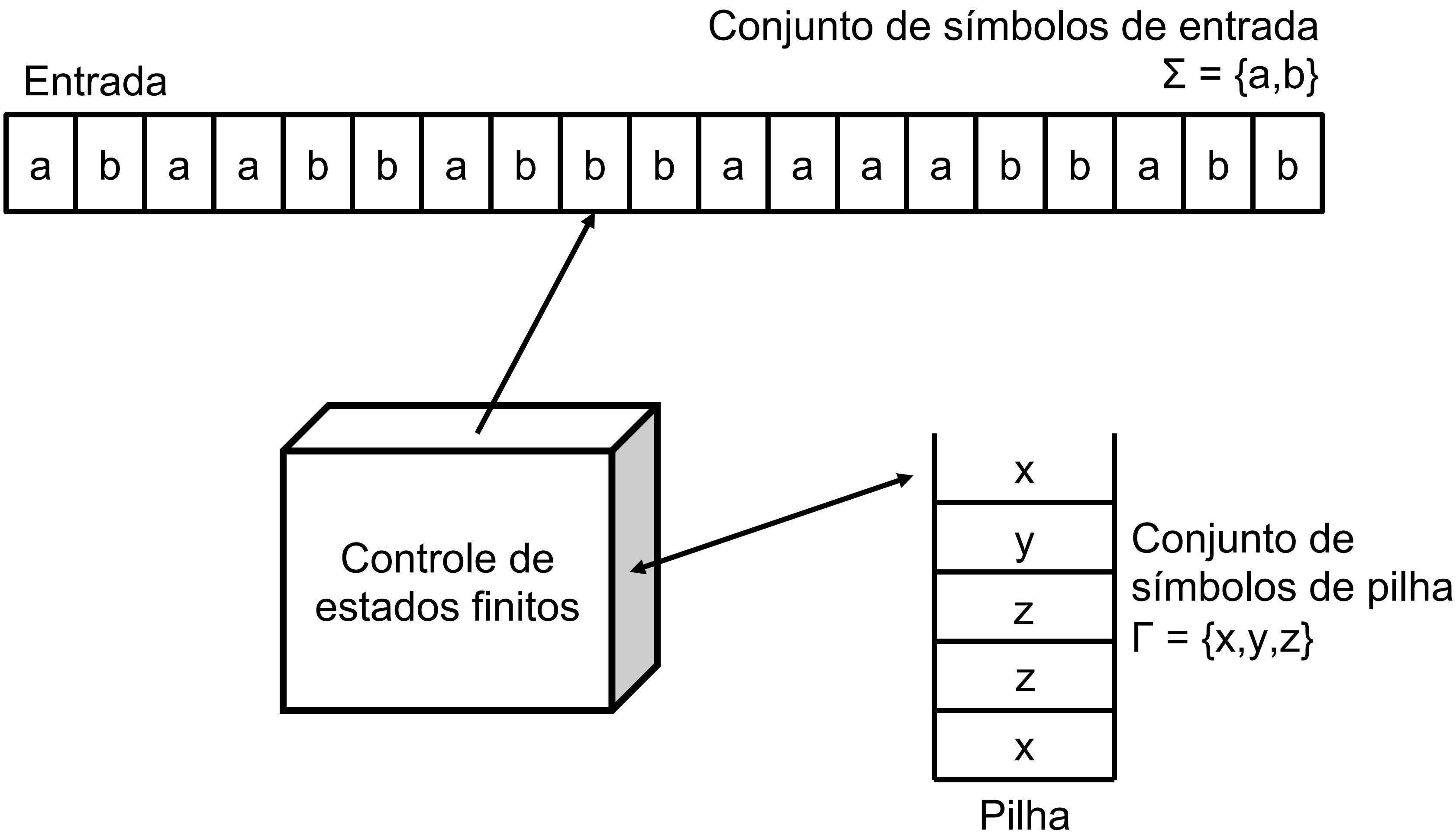
Autômatos de Pilha
(Pushdown Automata – PDA)

Autômatos de pilha

- é um ε-NFA com a inclusão de uma pilha

- Pilha = memória adicional
  - Além dos estados finitos
  - Quantidade infinita de informações
- A pilha pode ser lida, aumentada e diminuída apenas no topo
- Autômatos de pilha reconhecem todas as linguagens livres de contexto e apenas as linguagens livre de contexto.....

Autômatos de pilha
(possivel interpretação deste mecanismo ...)



Autômatos de pilha
(possivel interpretação deste mecanismo ...)

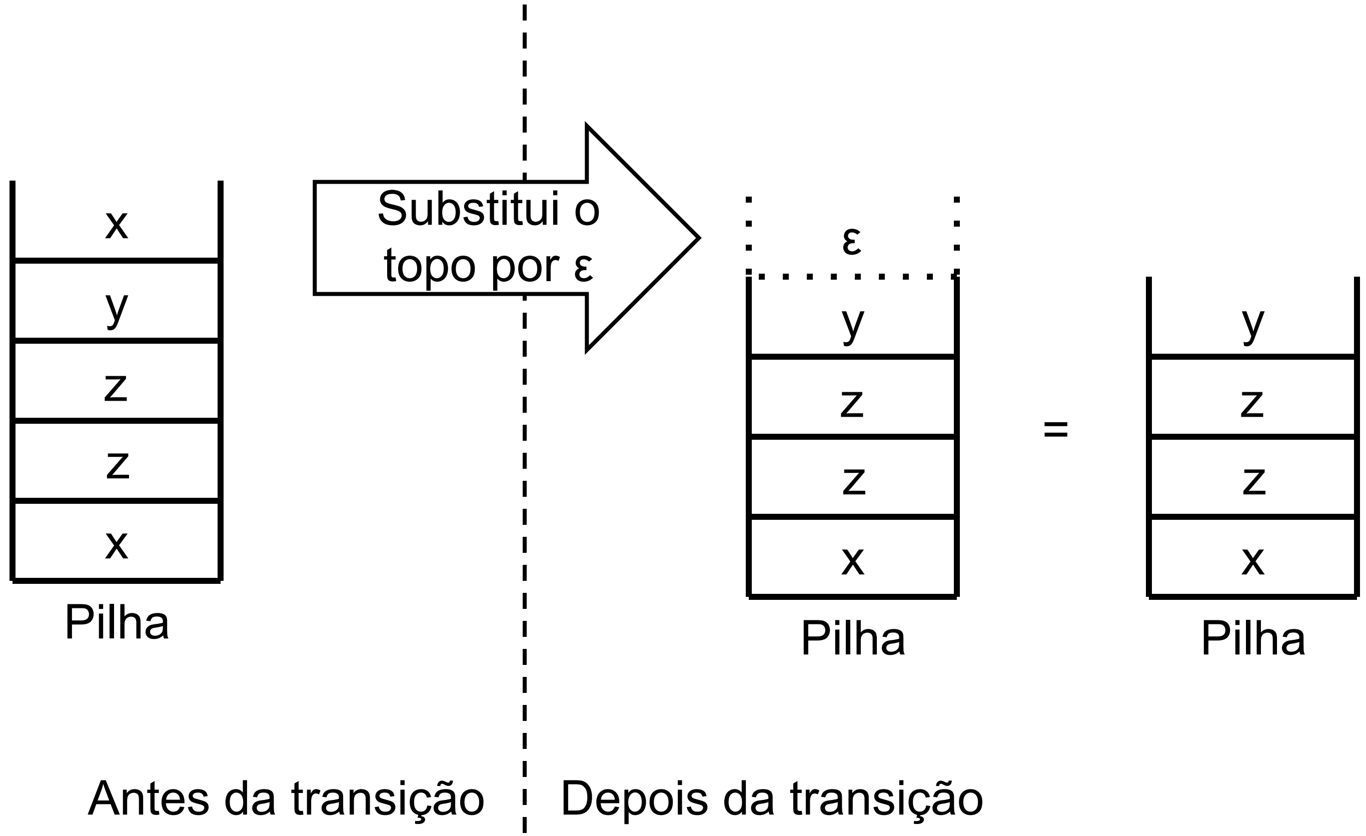
- O controle de estados finitos lê as entradas, um símbolo de cada vez
- O controle tem permissão para observar o símbolo no topo da pilha
  - Pode basear a transição:
    - em seu estado atual
    - no símbolo de entrada
    - no símbolo presente no topo da pilha
  - Opcionalmente, a entrada pode ser ε
    - Ou seja, podem haver transições “espontâneas”

Autômatos de pilha
(possivel interpretação deste mecanismo ...)

- Em uma transição, o autômato de pilha:
  - Consome da entrada o símbolo que utiliza na transição
    - Se for uma transição espontânea, nenhum símbolo de entrada é consumido
  - Vai para um novo estado, que pode ou não ser o mesmo estado anterior
  - Substitui o símbolo no topo da pilha por qualquer cadeia

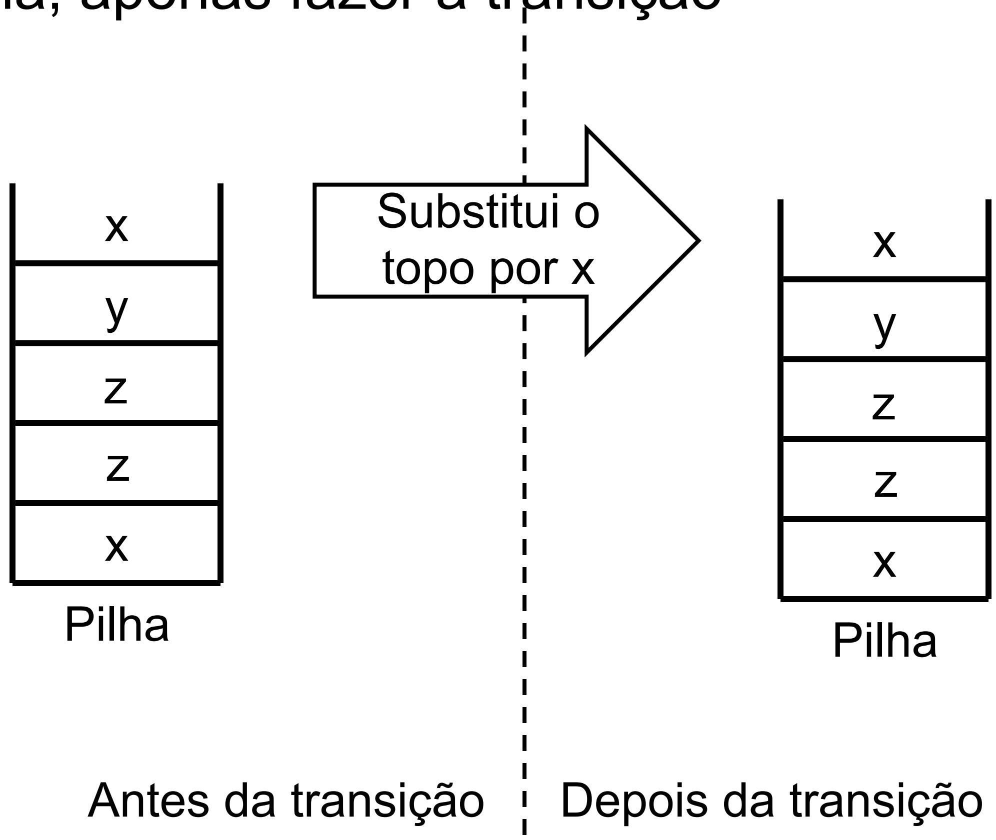
Autômatos de pilha

- Substituição de símbolo no topo da pilha
  - Se for ε, equivale a uma extração (pop) da pilha



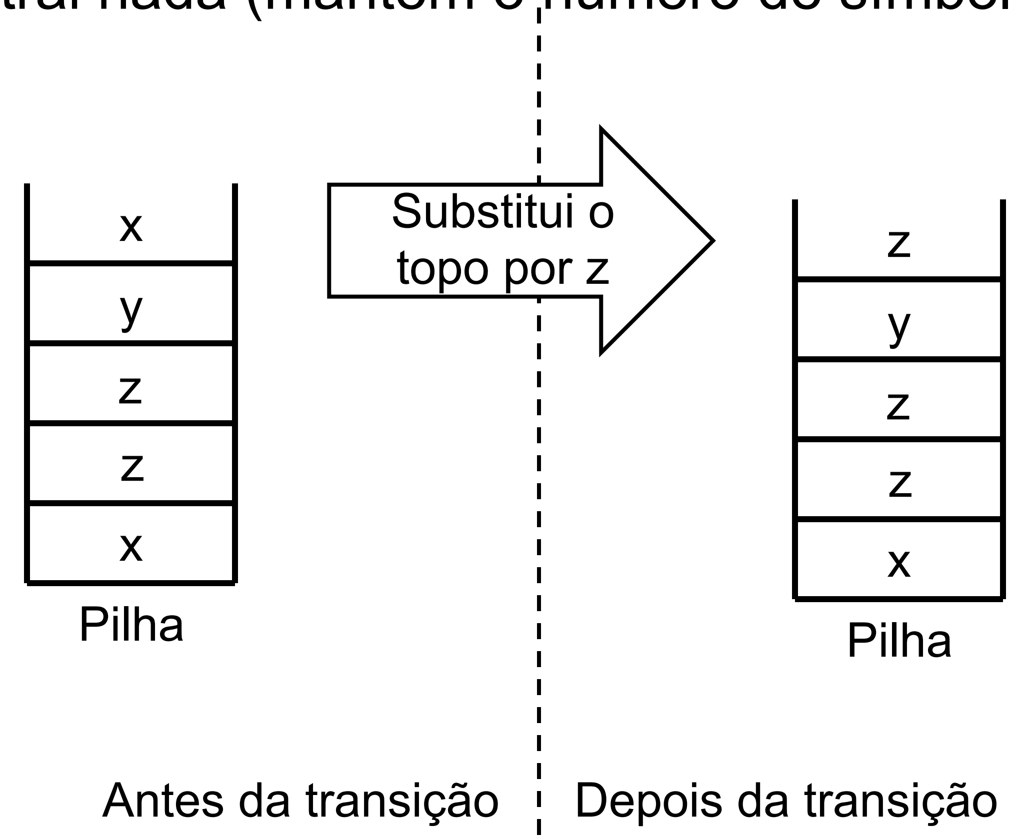
Autômatos de pilha

- Substituição de símbolo no topo da pilha
  - Se for o mesmo que já estava, equivale a não mudar a pilha, apenas fazer a transição



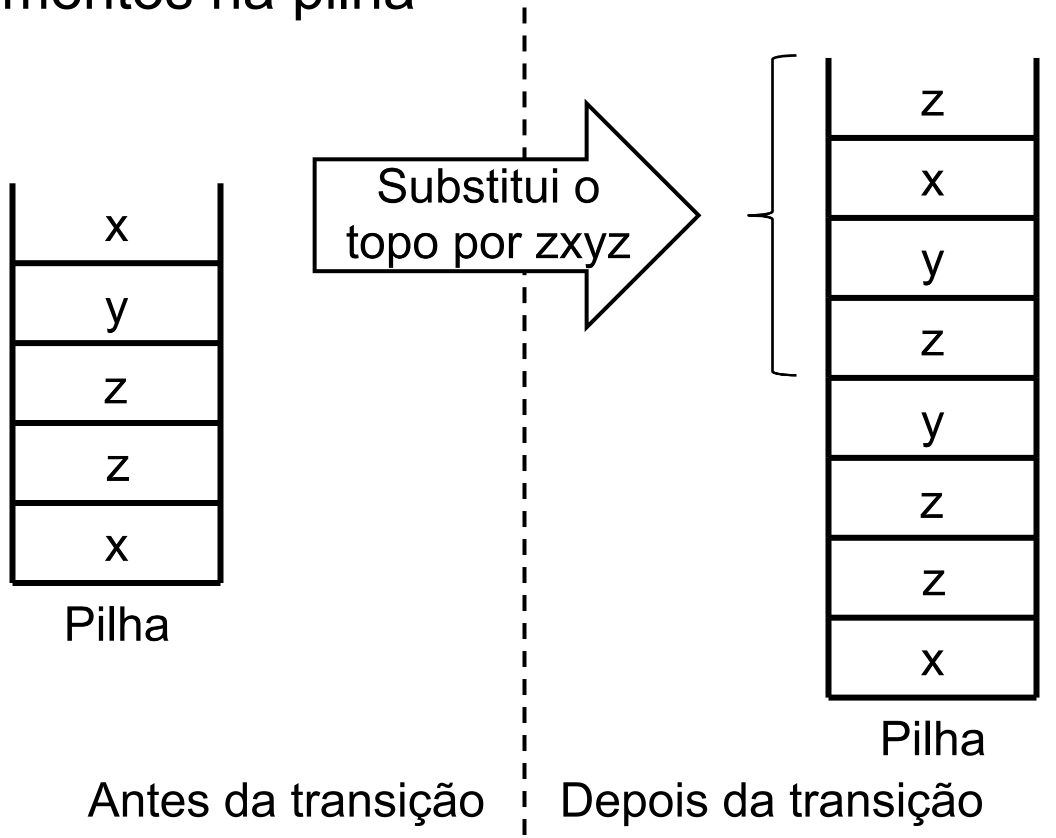
Autômatos de pilha

- Substituição de símbolo no topo da pilha
  - Se for outro símbolo, altera o topo, mas não insere nem extrai nada (mantém o número de símbolos na pilha)



Autômatos de pilha

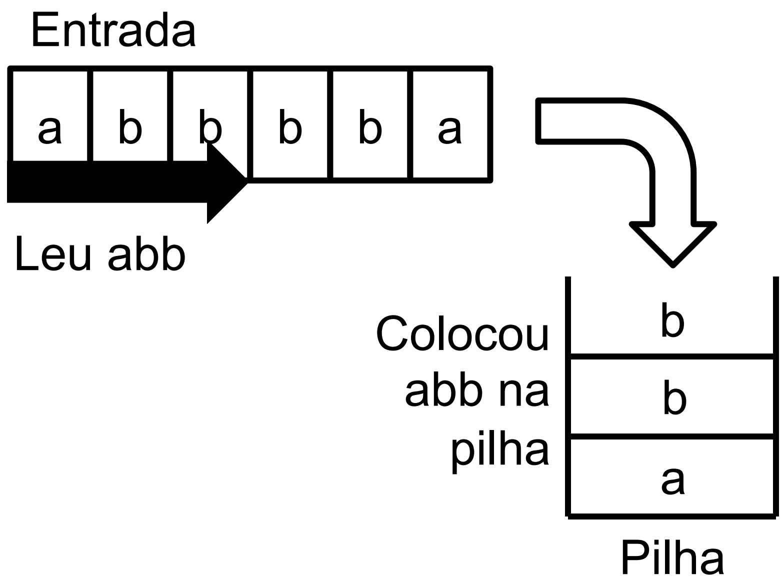
- Substituição de símbolo no topo da pilha
  - Se for uma cadeia com dois ou mais símbolos, insere elementos na pilha



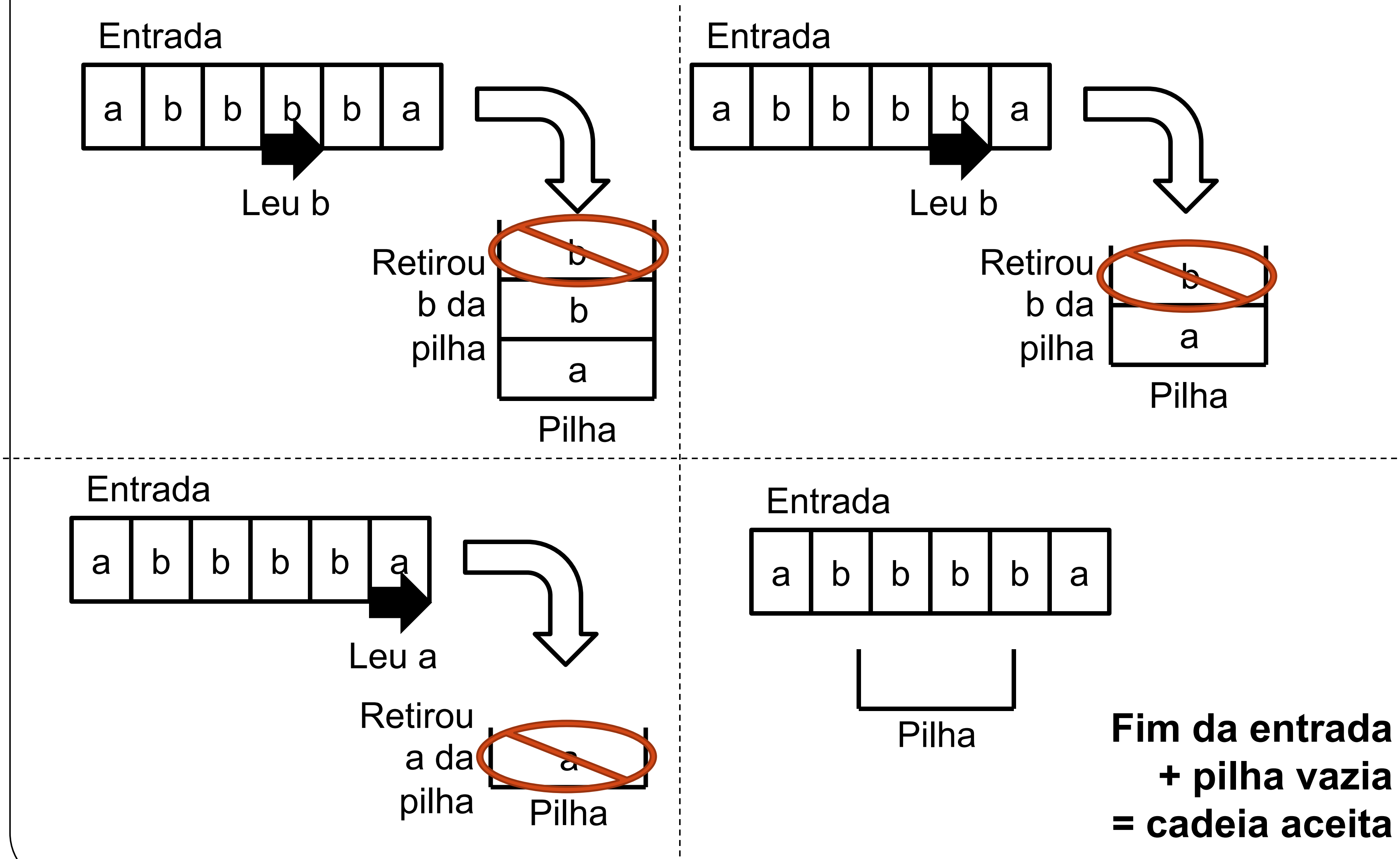
Autômatos de pilha

- Ex: $L = \{ww^R \mid w \text{ está em } \{0,1\}^*\}$
  - Três estados: q0, q1 e q2
  - q0 = ainda não chegou no meio da entrada
  - q1 = passou do meio da entrada
  - q2 = chegou no fim da entrada
- Funcionamento:
  - Começa em q0
  - Estando em q0, vai lendo a entrada e colocando uma cópia do símbolo lido no topo da pilha
  - No meio da pilha (usando o poder de oráculo do não-determinismo), muda para q1
  - Estando em q1, vai lendo a entrada. Compara-se o símbolo lido com o símbolo no topo da pilha. Se for igual, substitui o topo da pilha por ε
  - No final da entrada, se a pilha estiver vazia (topo = ε), aceita a cadeia

Autômatos de pilha



Autômatos de pilha



Fim da entrada
+ pilha vazia
= cadeia aceita

Autômatos de pilha

- Definição formal de um PDA (*PushDown Automata*)

$$\mathbf{P = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)}$$

- Onde:
  - Q = Um conjunto finito de estados
  - Σ = Um conjunto finito de símbolos de entrada
  - Γ = Um alfabeto de pilha finito – conjunto de símbolos que temos permissão para inserir na pilha (pode incluir elementos de Σ)
  - δ = Função de transição – governa o comportamento do autômato
  - q_0 = Estado inicial
  - Z_0 = Símbolo de início – Inicialmente, a pilha do PDA consiste em uma instância desse símbolo e em nada mais
  - F = Conjunto de estados de aceitação

Autômatos de pilha

- Função de transição
  - $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\varepsilon\} \rightarrow 2^{(Q \times \Gamma^*)}$
  - O argumento é uma tripla (q, a, X), onde:
    - q é um estado em Q
    - a é um símbolo de entrada em Σ ou a=ε (cadeia vazia)
    - X é um símbolo da pilha, isto é, um elemento de Γ
  - A saída de δ é um conjunto finito de pares (p,γ), onde:
    - p é o novo estado
    - γ é a cadeia de símbolos da pilha que substitui X no topo da pilha
      - Se γ = ε, a pilha é extraída
      - Se γ = X, a pilha fica inalterada
      - Se γ = YZ, X é substituído por Z e Y é inserido na pilha

Autômatos de pilha

- Exemplo: Vamos projetar um PDA P para aceitar a linguagem L = $\{ww^R \mid w$ está em $\{0,1\}^*\}$:
- $P = (\{q_0,q_1,q_2\}, \{0,1\}, \{0,1,\$\}, \delta, q_0, \$, \{q_2\})$
  - Obs: \$ é um símbolo que fica no fundo da pilha, inicialmente, e será usado para marcar quando a pilha está vazia
- Onde δ é definido pelas seguintes regras:

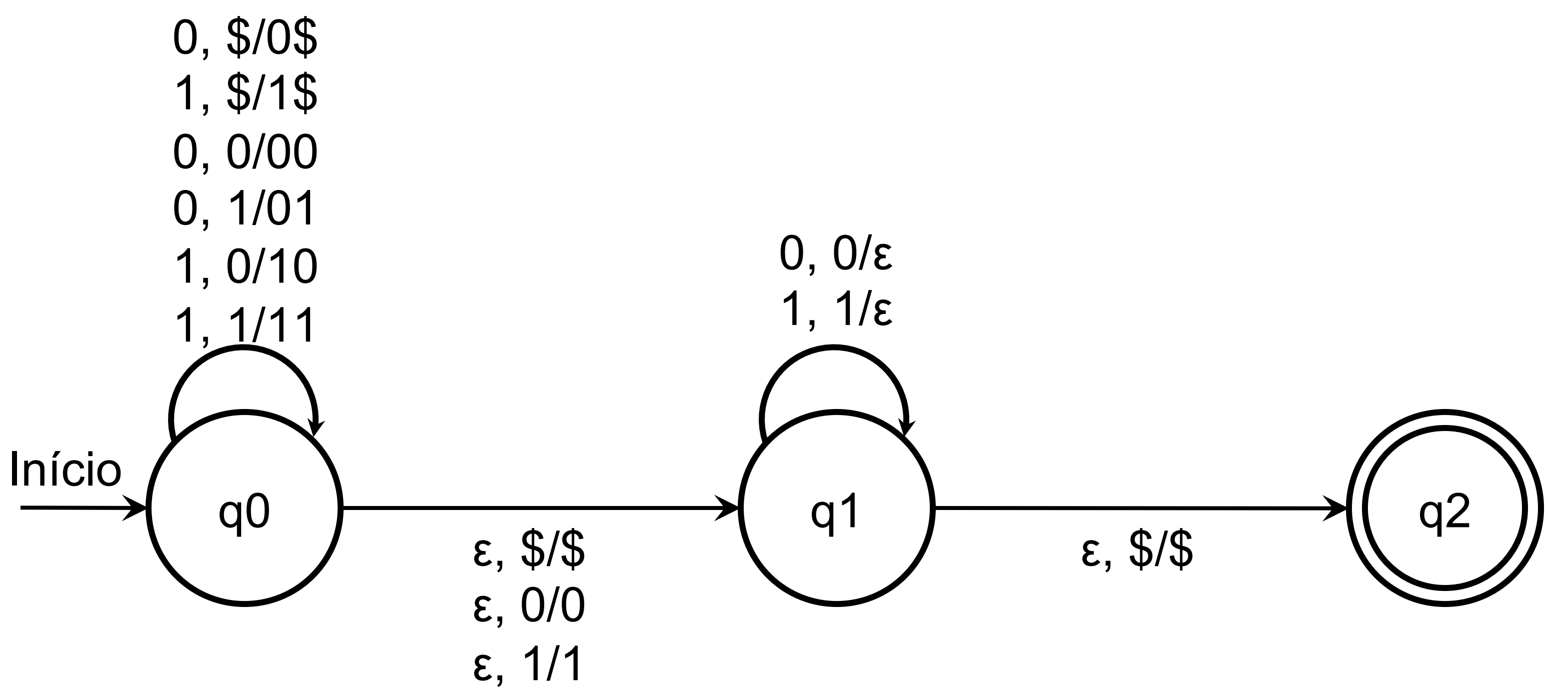
1. Empilhando
   - $\delta(q0,0,\$)=\{(q0,0\$)\}$ e $\delta(q0,1,\$)=\{(q0,1\$)\}$
   - $\delta(q0,0,0)=\{(q0,00)\}$, $\delta(q0,0,1)=\{(q0,01)\}$, $\delta(q0,1,0)=\{(q0,10)\}$ e $\delta(q0,1,1)=\{(q0,11)\}$
2. Adivinhando o meio da cadeia
   - $\delta(q0,\varepsilon,\$)=\{(q1,\$)\}$, $\delta(q0,\varepsilon,0)=\{(q1,0)\}$ e $\delta(q0,\varepsilon,1)=\{(q1,1)\}$
3. Desempilhando
   - $\delta(q1,0,0)=\{(q1,\varepsilon)\}$ e $\delta(q1,1,1)=\{(q1,\varepsilon)\}$
4. Checando a pilha vazia
   - $\delta(q1,\varepsilon,\$)=\{(q2,\$)\}$

Autômatos de pilha

- A lista de transições para um PDA nem sempre é fácil de acompanhar
- Uma forma melhor é um diagrama de transição para PDAs:
  - Os nós correspondem aos estados do PDA
  - Uma seta identificada por Início indica o estado inicial, e estados com círculos duplos são estados de aceitação
  - Os arcos correspondem a transições do PDA
    - Mas com algumas extensões, conforme a seguir

Autômatos de pilha

- Um arco é identificado por a, X/α
  - a = símbolo de entrada
  - X = símbolo no topo da pilha
  - α = cadeia a substituir o topo da pilha



Autômatos de pilha

- Outra forma é criar uma tabela tridimensional

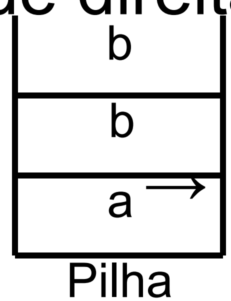
Entrada:	0			1			ε		
Pilha:	0	1	$	0	1	$	0	1	$
q0	{(q0,00)}	{(q0,01)}	{(q0,0$)}	{(q0,10)}	{(q0,11)}	{(q0,1$)}	{(q1,0)}	{(q1,1)}	{(q1,$)}
q1	{(q1,ε)}	Ø	Ø	Ø	{(q1,ε)}	Ø	Ø	Ø	{(q2,$)}
q2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Autômatos de pilha

- Para facilitar o acompanhamento da execução de um autômato, existe também o conceito de configuração (ou descrição) instantânea
  - Um texto que resume o estado da execução em um determinado momento, com as informações essenciais
    - Estado atual do PDA
    - Entrada a ser lida
    - Conteúdo da pilha

Autômatos de pilha

- A configuração instantânea (CI) de um PDA é representada por uma tripla (q, w, γ), onde:
  - q é o estado
  - w é a parte restante da entrada
  - γ é o conteúdo da pilha
- Convencionalmente, mostramos o topo da pilha na extremidade esquerda e a parte inferior na extremidade direita
  - Ex:



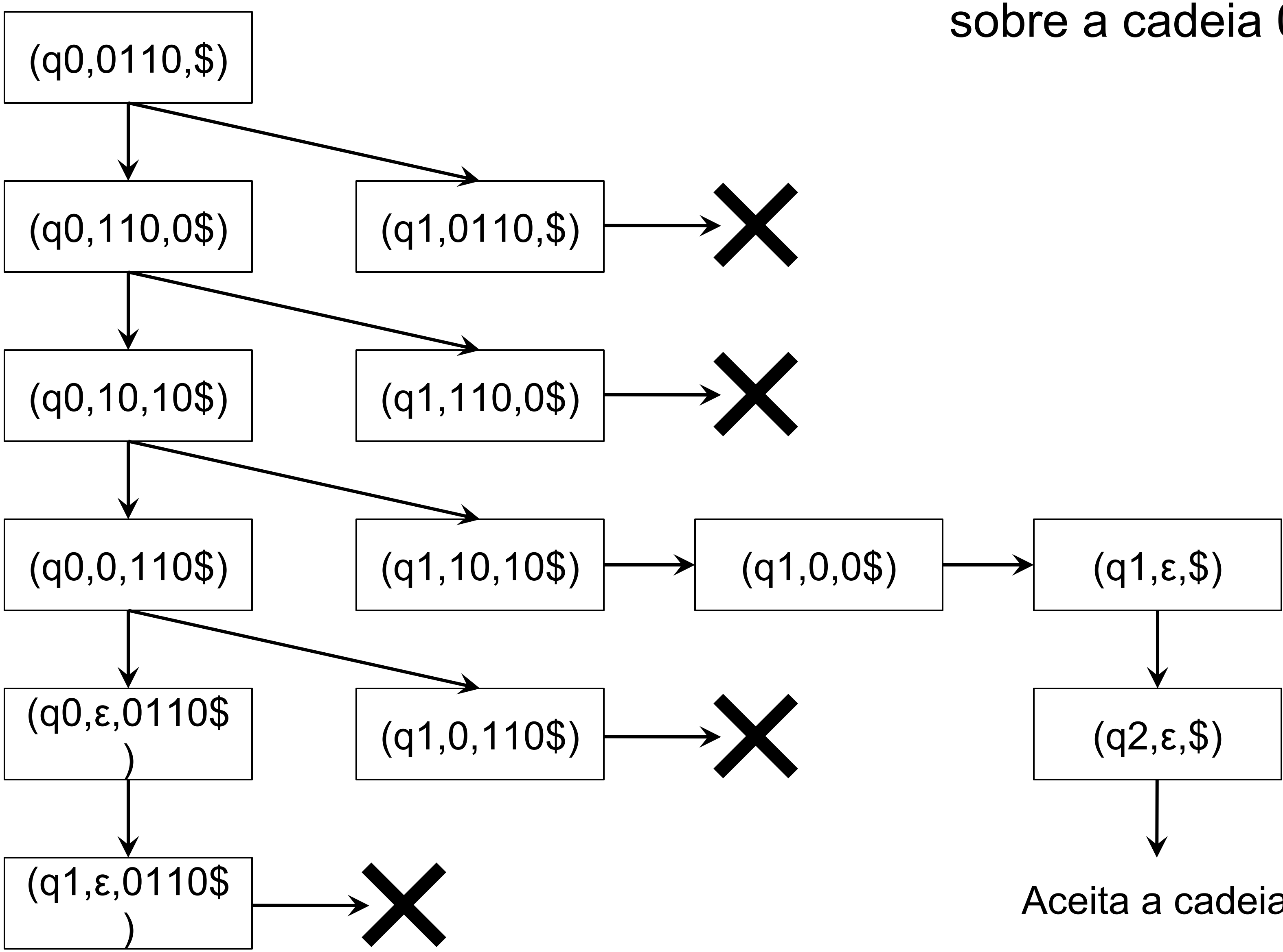
γ = bba

Autômatos de pilha

- Formalmente: um movimento genérico em um PDA é descrito pelas seguintes configurações instantâneas
  - $(q,aw,X\beta) \vdash (p,w,\alpha\beta)$
- Supondo que $\delta(q,a,X)$ contém (p,α)
  - A notação $\overset{*}{\vdash}$ indica uma sequência de movimentos

Autômatos de pilha

Exemplo: o autômato apresentado anteriormente sobre a cadeia 0110



Autômatos de pilha

- Exercício: mostre as sequências de configurações instantâneas para a cadeia 1111

Autômatos de pilha

- As linguagens de um PDA
- Existem duas abordagens para decidir se um PDA aceita ou não uma entrada
  - Aceitação pelo estado final
  - Aceitação por pilha vazia
- Ambas são equivalentes
  - Uma linguagem L tem um PDA que a aceita pelo estado final se e somente se L tem um PDA que a aceita por pilha vazia
  - Ou seja, é possível converter um PDA que aceita L por estado final em outro PDA que aceita L por pilha vazia
    - E vice-versa

Autômatos de pilha

- Aceitação por estado final
  - Seja $P = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ um PDA
    - Então L(P), a linguagem aceita por P pelo estado final, é:
    - $L(P) = \{w \mid (q_0,w,Z_0) \vdash^* (q,\varepsilon,\alpha)\}$
    - Para algum estado q em F e qualquer cadeia de pilha α
- Ou seja, é o conjunto de todas as cadeias que o PDA pode processar, a partir da configuração instantânea inicial, consumindo todos os símbolos da cadeia, que chegam a um estado de aceitação
  - Não interessa o que "sobrar" na pilha

Autômatos de pilha

- Aceitação por pilha vazia
  - Seja $P = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ um PDA
    - Então N(P), a linguagem aceita por P por pilha vazia, é:
    - $N(P) = \{w \mid (q_0,w,Z_0) \vdash^* (q,\varepsilon,\varepsilon)\}$
    - Para qualquer estado q (não necessariamente em F)
- Ou seja, é o conjunto de cadeias que o PDA pode processar, a partir da configuração instantânea inicial, consumindo todos os símbolos da cadeia e deixando a pilha vazia no final
  - Não interessa em qual estado o autômato parou
  - Portanto, quando a aceitação é por pilha vazia, a descrição pode omitir o último elemento da tupla:
    - $P = (Q,\Sigma,\Gamma,\delta,q_0,Z_0)$

Autômatos de pilha

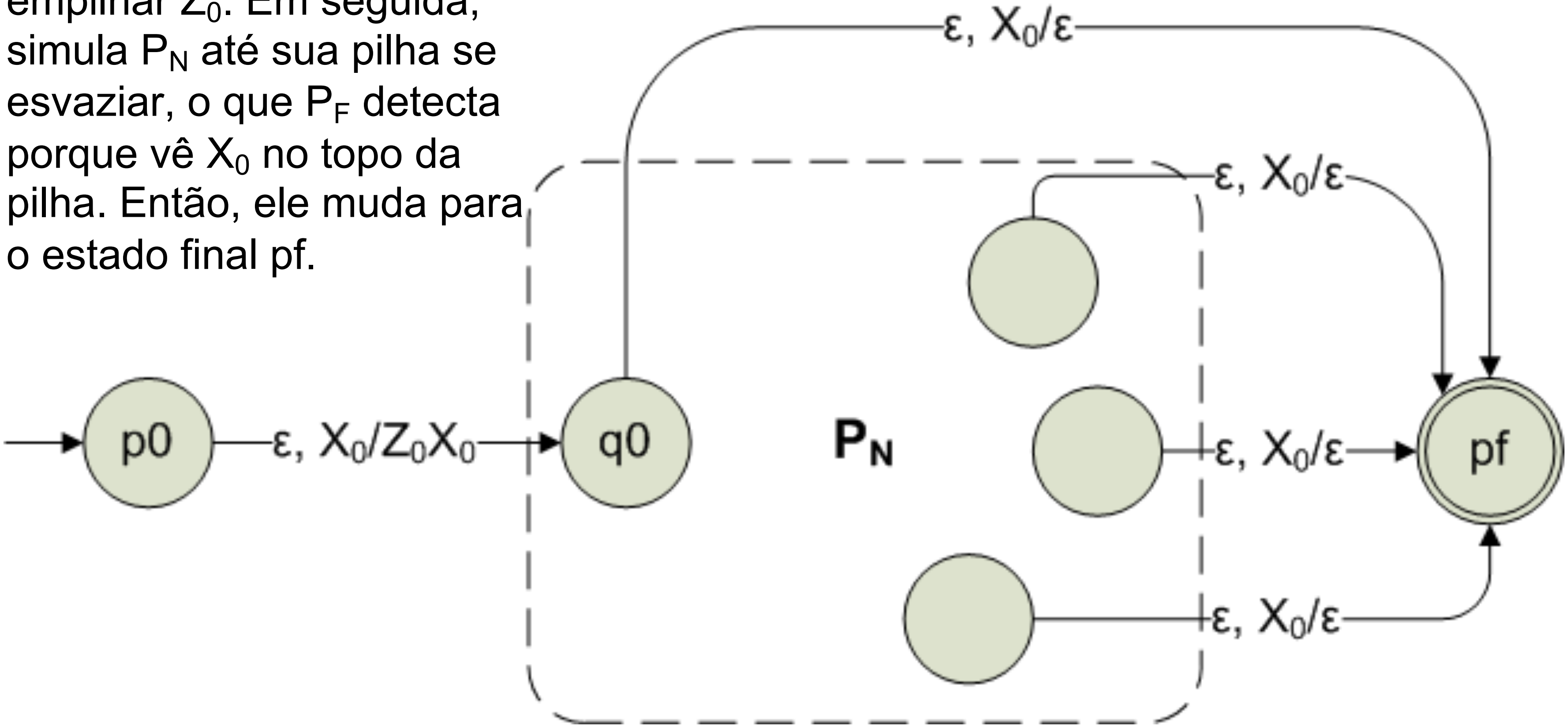
- Aceitação por pilha vazia = Aceitação por estado final
- Prova: por construção
  - Conversão de pilha vazia → estado final
  - Conversão de estado final → pilha vazia

De pilha vazia ao estado final

- Dado um PDA $P_N = (Q,\Sigma,\Gamma,\delta_N,q0,Z_0,F)$ que aceita por pilha vazia:
- Criaremos um PDA P_F que aceita por estado final:
  - $P_F = (Q \cup \{p0, pf\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p0, X_0, \{pf\})$
  - Novo símbolo X_0
  - Novo estado inicial p0
  - Novo estado final pf
  - Novas transições (δ_F) conforme a seguir

De pilha vazia ao estado final

P_F começa com X_0. O primeiro movimento é empilhar Z_0. Em seguida, simula P_N até sua pilha se esvaziar, o que P_F detecta porque vê X_0 no topo da pilha. Então, ele muda para o estado final pf.

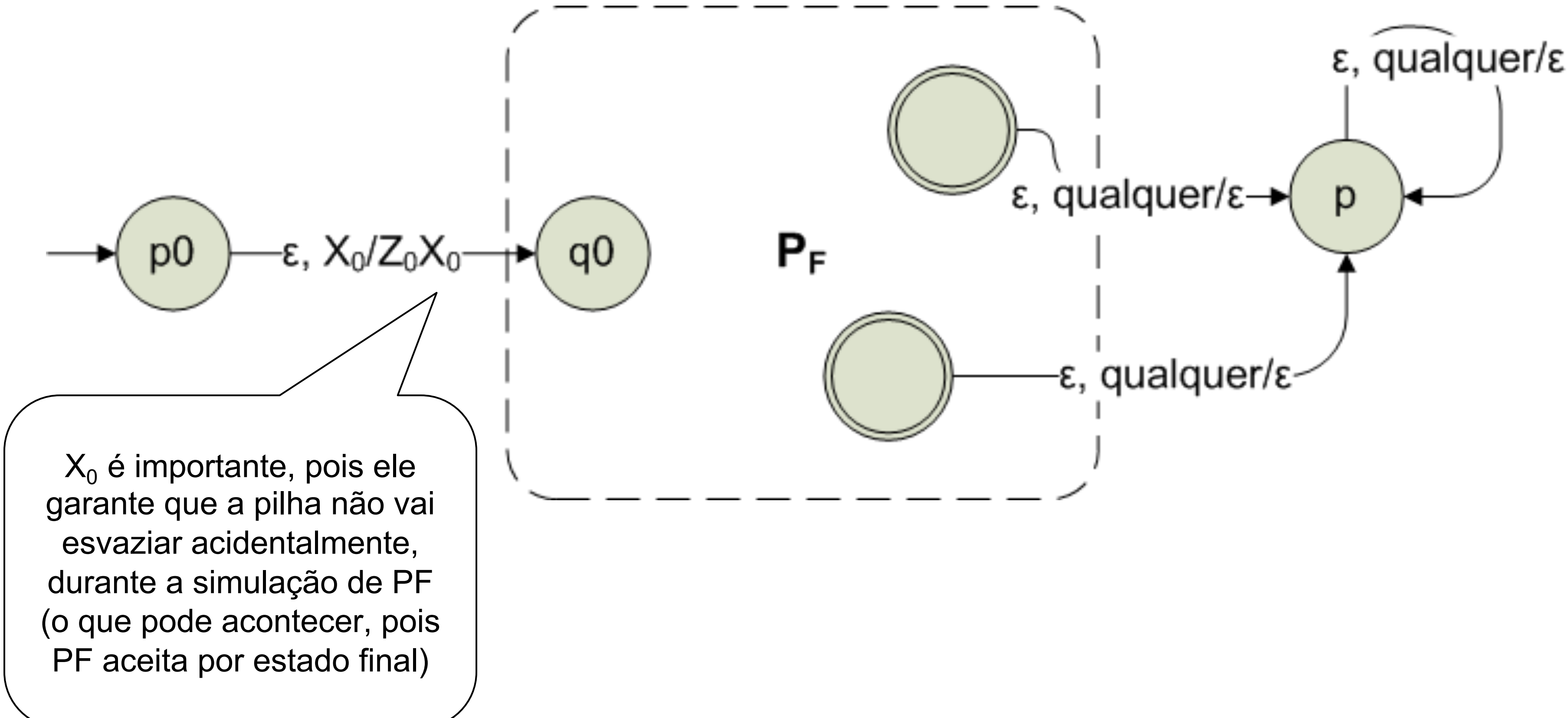


De estado final para pilha vazia

- Dado um PDA $P_F = (Q,\Sigma,\Gamma,\delta_F,q0,Z_0,F)$ que aceita por estado final:
- Criaremos um PDA P_N que aceita por pilha vazia:
  - $P_F = (Q \cup \{p0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p0, X_0)$
  - Novo símbolo X_0
  - Novo estado inicial p0
  - Novo estado p (que representa a pilha vazia)
  - Novas transições (δ_N) conforme a seguir

De estado final para pilha vazia

P_N começa com X_0. Inicialmente, empilha Z_0 e simula P_F até chegar a um estado de aceitação. Então, ele muda para um estado p que nada faz, a não ser esvaziar a pilha.



Autômatos de pilha

- Se uma linguagem é L(P) ou N(P) para algum PDA, então L é livre de contexto
  - Prova: por construção
    - Conversão de CFG → PDA
      - (Essencialmente, é o conteúdo da disciplina de compiladores, não vamos cobrir aqui)
    - Conversão de PDA → CFG
      - (Não tem muita utilidade prática)

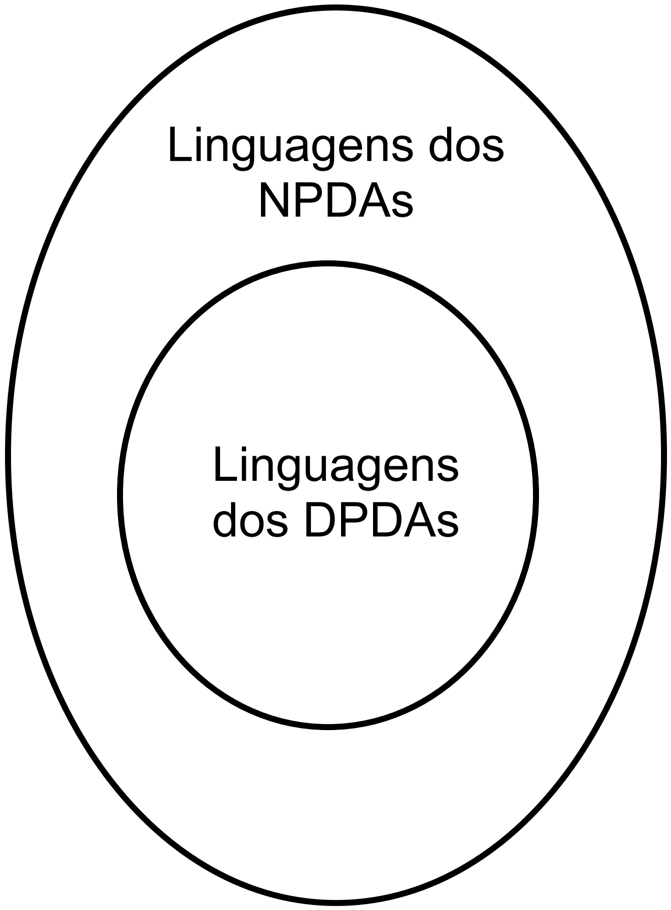
Determinismo em PDAs

Determinismo em PDAs

- O não-determinismo é um bom “truque de programação”, pois ajuda a projetar linguagens/autômatos
  - Para autômatos finitos, o não-determinismo não dá poder aos autômatos, ou seja:
    - NFAs reconhecem as mesmas linguagens que DFAs
    - É possível converter NFAs em DFAs e vice-versa
  - A escolha de quando converter (em tempo de execução ou pré-execução) fica a cargo do projetista, e depende do cenário de aplicação

Determinismo em PDAs

- Para autômatos de pilha, o mesmo não acontece
- O não-determinismo AUMENTA a capacidade reconhecedora de um PDA
  - Ou seja, PDAs determinísticos (DPDAs) reconhecem menos linguagens do que PDAs não-determinísticos (NPDAs)
    - Equivalente a dizer que existem linguagens reconhecidas por NPDAs que não são reconhecidas por DPDAs



PDA determinístico - DPDA

- Definição informal é a mesma do que para autômatos finitos determinísticos
  - Ou seja, em um DPDA, nunca há alternativa de escolha para movimento, e sempre o autômato sabe o que fazer
  - Isso significa que, para toda combinação de entrada, estado e topo da pilha, há uma e somente uma possibilidade de transição
- Importante: a transição vazia (ε) aqui não é indicativo de não-determinismo!!
  - Pois pode haver uma decisão com base no topo da pilha.
  - O problema é haver conflito entre consumir a entrada ou não!

PDA determinístico - DPDA

- O caso da entrada vazia
  - Para um determinado topo da pilha X, e uma entrada a
  - O DPDA:
    - Ou define uma transição com base em a (e a transição com base em ε fica vazia)
    - Ou define uma transição com base em ε (e a transição com base em a fica vazia)
- Na tabela, as células correspondentes às colunas ε nunca sobrepõem o que foi definido nas células das colunas das entradas

Propriedades das Linguagens Livre de Contexto

Propriedades das Linguagens Livre de Contexto

- Embora as Linguagens Livres de Contexto sejam mais gerais que as Regulares, ainda são relativamente restritas.
- Mas como é possível determinar se uma Linguagem é Livre de Contexto ? Como verificar se uma Linguagem Livre de Contexto é infinita ou finita (ou até mesmo vazia) ?

Lema do bombeamento para linguagens livres de contexto

Lema do bombeamento

- Para linguagens regulares
  - Prova que linguagens não são regulares
  - Base: para linguagens regulares, temos autômatos finitos
    - Autômatos finitos não conseguem contar
    - Esse é o limite das linguagens regulares, e basicamente é o que o lema busca provar
- Para linguagens livres de contexto
  - Prova que linguagens não são livres de contexto
  - Base: para linguagens livres de contexto, temos autômatos de pilha (veremos a seguir)
    - Autômatos de pilha conseguem contar somente uma “coisa”, mas não duas, ao mesmo tempo
    - Esse é o limite das linguagens livre de contexto, e basicamente é o que o lema busca provar

Lema do bombeamento

- Parecido com o lema do bombeamento para linguagens regulares
  - Em LR
    - Dividimos uma cadeia s em 3 partes, xyz
    - E “bombeamos” y (isto é, fazemos xy^iz para $i \geq 0$), e a cadeia resultante ainda deve estar na linguagem
  - Para LLC
    - Dividimos uma cadeia s em 5 partes, uvwxy
    - E “bombeamos” v e x (isto é, fazemos uv^iwx^iy para $i \geq 0$), e a cadeia resultante ainda deve estar na linguagem
- Ex: Se abcdefg faz parte da linguagem
  - Fazemos s=uvwxy, u=a,v=bc,w=de,x=f,y=g
  - Então “bombeando” v e x zero ou mais vezes, sempre obtemos cadeias que fazem parte da linguagem
  - Ou seja, adeg (i=0), abcbcdeffg (i=2) e abcbcbcdefffg (i=3) fazem parte da linguagem
    - Assim como todas as outras cadeias com v e x “bombeadas”

Lema do bombeamento para linguagens livres de contexto

- Seja L uma linguagem livre de contexto.
  - Então, existe uma constante n tal que, se z é qualquer cadeia em L tal que |z| é pelo menos n, podemos escrever z = uvwxy, sujeito às seguintes condições:
    - $|vwx| \leq n$. Ou seja, a porção intermediária não é muito longa.
    - $vx \neq \varepsilon$. Tendo em vista que v e x são os fragmentos a serem "bombeados", essa condição diz que pelo menos uma das cadeias que bombeamos não deve ser vazia.
    - Para todo $i \geq 0$, uv^iwx^iy está em L. Isto é, as duas cadeias v e x podem ser "bombeadas" qualquer número de vezes, incluindo 0, e a cadeia resultante ainda será um elemento de L.

Linguagens não livres de contexto

- Linguagens que precisam "contar duas coisas"
  - Ex: correspondência entre três grupos de símbolos de igualdade
    - $\{0^n1^n2^n \mid n \geq 1\}$ (ou $0^+1^+2^+$ com um número igual de cada símbolo)
    - Exs: 012, 001122, 000111222
  - Ex: comparar dois pares com números iguais de símbolos, quando os pares se intercalam
    - $\{0^i1^j2^i3^j \mid i \geq 1 \text{ e } j \geq 1\}$
    - Exs: 00012223, 00111112233333

Linguagens não livres de contexto

- Exemplo importante
  - Linguagens livres de contexto não podem comparar duas cadeias de comprimento arbitrário, se as cadeias forem escolhidas a partir de um alfabeto com mais de um símbolo.
  - Seja L = {ww | w está em {0,1}*}. Isto é, L consiste em cadeias repetitivas, como ε, 0101, 00100010, 110110.
  - Usando o lema do bombeamento, é possível provar que L não é livre de contexto

Aplicando o lema do bombeamento para linguagens livres de contexto

- Se L é livre de contexto, então seja n a constante de seu lema do bombeamento
  - Considere a cadeia $z = 0^n1^n0^n1^n$
  - Essa cadeia é 0^n1^n repetida, e assim, z está em L
- Vamos desmembrar z = uvwxy, tal que $|vwx| \leq n$ e $vx \neq \varepsilon$
  - É possível mostrar que uwy não está em L
    - Não faremos essa prova aqui!
  - Ou seja, "bombeando" v e x zero vezes, obtemos uma cadeia que não está em L
    - Ferindo o lema
  - Isso é uma contradição, e portanto concluímos que L não é livre de contexto

Linguagens não livres de contexto

- Esse caso é particularmente interessante
- Considere a seguinte cadeia, em uma linguagem de programação típica:

Irá acusar erro aqui, pois a variável nmero não foi declarada

```
String numero = 0;
if (nmero > 0) {
  System.out.println("Nunca vai entrar
    aqui");
}
```

- Em algumas LP, variáveis precisam ser declaradas antes de serem utilizadas
  - É o mesmo caso da linguagem {ww | w em um alfabeto com mais de um símbolo}

Linguagens não livres de contexto

- Outros exemplos: declaração de pacotes, macros, chamada de funções, etc.
- Ou seja, gramáticas livres de contexto não conseguem impor todas as restrições “semânticas” de uma linguagem de programação típica
- Como é feito então?
  - Outros mecanismos, como uma “tabela de símbolos”
  - Mais sobre isso na disciplina de compiladores

Autômatos a Pilha e Gramáticas Livre de Contexto

Linguagens livres de contexto

- A partir de uma gramática livre de contexto é possível obter um autômato a pilha que reconhece a mesma linguagem

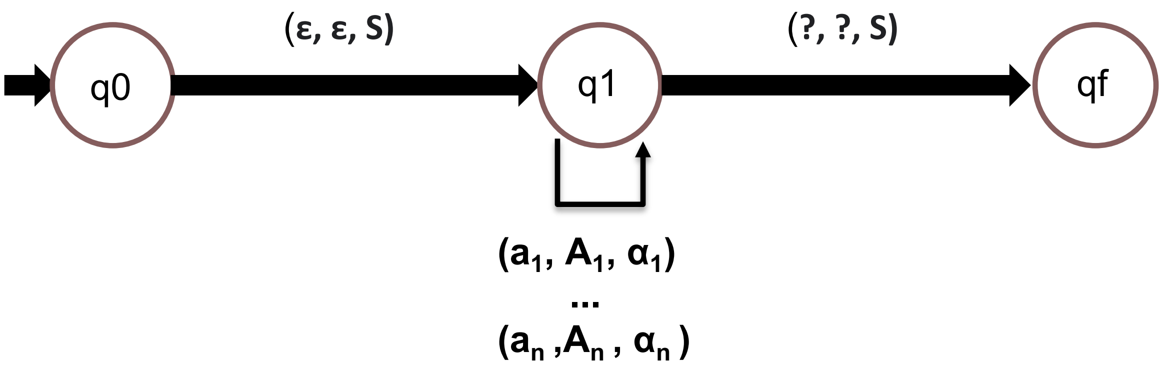
- A partir de um autômato a pilha é possivel obter uma gramática livre de contexto que gera a mesma linguagem reconhecida pelo autômato a pilha.

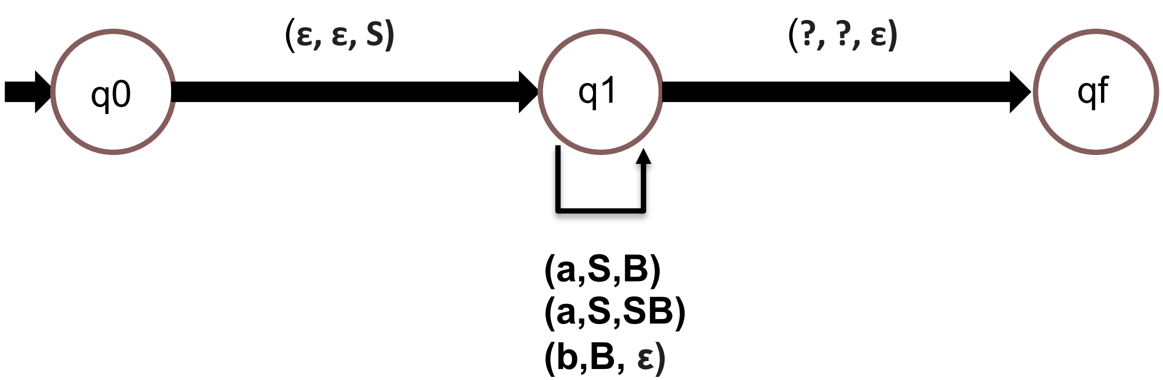
Linguagens livres de contexto

- Se L é uma livre de contexto então

Existe um autômato a pilha que reconhece a linguagem L e esse autômato pode ter apenas um estado.

Exemplo de Autômatos a Pilha e Gramáticas Livre de Contexto





Propriedades das Linguagens Livre de Contexto

Linguagens livres de contexto

- É fechada sob as operações de:

- União
- Concatenação

- Não é fechada sob a operação de

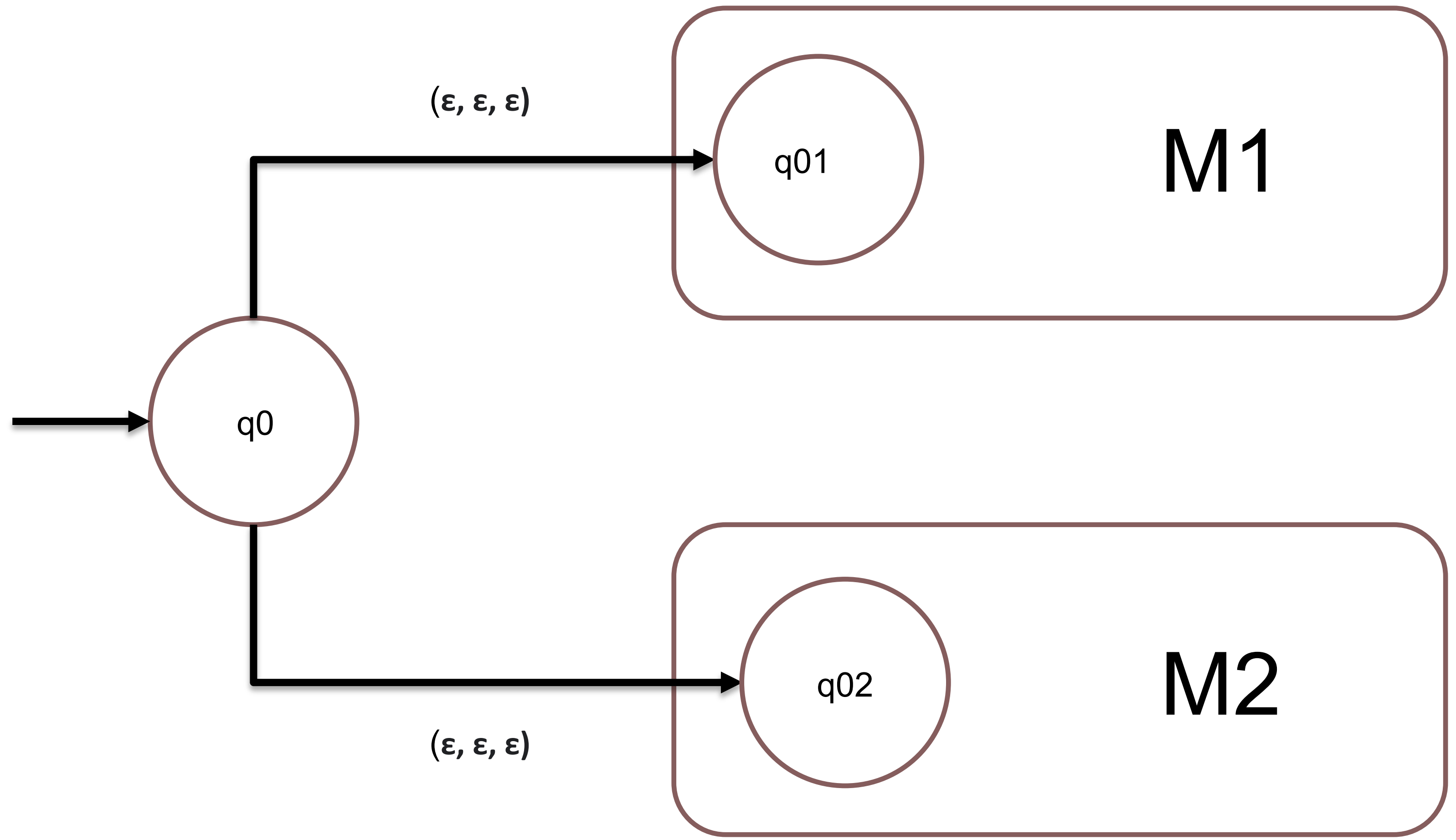
- Intersecção

Operação : União

- Suponha L1 e L2, LLC. Então existem Automatos com Pilha :
  - M1 = ($\sum 1$, Q1,$^{\delta}$1,q01,F1,V1)
  - M2 = ($\sum 2$, Q2,$^{\delta}$2,q02,F2,V2)
- Tais que ACEITA(M1) = L1 e ACEITA(M2) = L2. Seja M3 a união dos autômatos M1 e M2 (suponha que Q1 INTERSECÇÃO Q2 INTERSECÇÃO {q0} = VAZIO e V1 INTERSECÇÃO V2 = VAZIO
  - M3 = ($\sum 1$ U $\sum 2$, Q1 U Q2 U {q0}, $^{\delta}$3,q0,F1 U F2, V1 U V2
  - Claramente, M3 reconhece L1 U L2

Operação : Concatenação

- Suponha L1 e L2, LLC. Então existem gramáticas Livres de Contexto :
  - G1 = (V1,T1,P1,S1)
  - G2 = (V2,T2,P2,S2)
- Tais que GERA(G1) = L1 e GERA(G2) = L2. Seja G3 = (V1 UNIAO V2 UNIAO {S}, T1 UNIAO T2, P1 UNIAO P2 UNIAO {S -> S1S2},S)
- Como a única produção G3 de S é S -> S1S2, claramente qualquer palavra gerada por G3 terá, como prefixo, uma palavra de L1 e, como sufixo, uma palavra de L2. Logo L1L2 é LLC



(ε, ε, ε)
q01
M1
q0
q02
M2
(ε, ε, ε)

Investigação se uma Linguagem livre de contexto é Vazia, Finita ou Infinita

- É possivel identificar (por um algoritmo) que uma particular linguagem livre de contexto L é

- Vazia
- Finita
- Infinita

Investigação se uma Linguagem livre de contexto é Vazia, Finita ou Infinita

- **Vazia.** Seja G = (V,T,P,S), GLC tal que GERA(G) = L. Seja G' = (V',T',P',S) equivalente a G, eliminando os símbolos inúteis. Se P' for vazio, então L é vazia

Investigação se uma Linguagem livre de contexto é Vazia, Finita ou Infinita

- **Finita e Infinita.** Seja G = (V,T,P,S), GLC tal que GERA(G) = L. Seja G' = (V',T',P',S) equivalente a G na Forma Normal de Chomsky. Se existe A, variável de V' tal que :
  - A-> BC, ou seja, A é referenciada no lado direito de alguma produção que não gera diretamente terminais;
  - X-> YA ou X-> AY, ou seja, se A é referenciada no lado direito de alguma produção;
  - Existe um ciclo em A do tipo $=>^{+}$alfabeta
  - Então A é capaz de gerar palavras de qualquer tamanho e, consequentemente a linguagem é infinita. Caso não exista tal A, então a linguagem é finita

Reconhecedores de Cadeias para Linguagens Livre de Contexto

Algoritmos de reconhecimento das LLCs

Classificados em:

Top-Down ou Preditivo

- constroi uma árvore de derivação a partir da raiz

(símbolo inicial da gramática) com ramos em direção às folhas (terminais)

Bottom-Up

- parte das folhas construindo a árvore de derivação em direção à raiz

Autômatos de pilha como Reconhecedor

- construção relativamente simples e imediata
- relação quase direta entre produções e as transições do AP
- algoritmo é:

– top-down

– simula derivação mais a esquerda

– não-determinismo são as produções alternativas da gramática

Autômatos de pilha como Reconhecedor

a partir de uma Gramática na Forma Normal de Greibach

– cada produção gera exatamente um terminal

– geração de w envolve |w| etapas de derivação

Cada variável pode ter diversas produções associadas

– AP testa as diversas alternativas

– número de passos para reconhecer w é proporcional a k |w|

– aproximação de k: metade da media de produções das variáveis.portanto, o AP construído

– tempo de reconhecimento proporcional ao expoente em |w|

– pode ser muito ineficiente para entradas mais longas

Autômato com Pilha Descendente

forma alternativa de construir AP, igualmente simples e com o mesmo nível de eficiência, a partir de uma GLC sem recursão a esquerda

simula a derivação mais a esquerda

- Algoritmo:

– inicialmente, empilha o símbolo inicial

– topo = variável: substitui, (não-determinismo), por todas as produções da variável

– topo = terminal: testa se e igual ao próximo simbolo da entrada

Autômato com Pilha Descendente

GLC $G = (V, T, P, S)$, sem recursão a esquerda

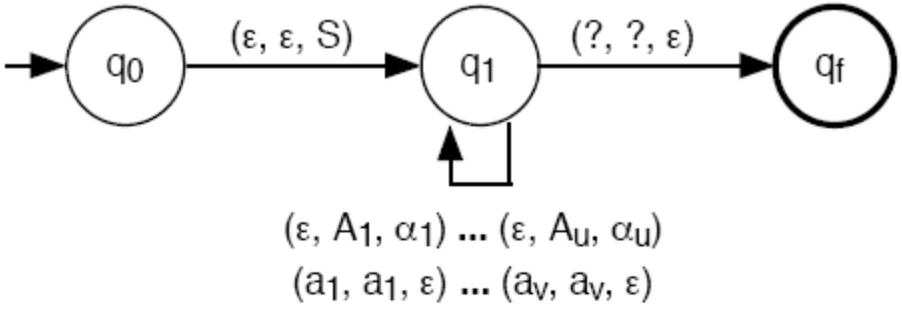
$M = (T, \{ q0, q1, qf \}, \delta, q0, \{ qf \}, V \cup T)$

$\delta(q0, \varepsilon, \varepsilon) = \{ (q1, S) \}$

$\delta(q1, \varepsilon, A) = \{ (q1, \alpha) \mid A \rightarrow \alpha \in P \}$ A de V

$\delta(q1, a, a) = \{ (q1, \varepsilon) \}$ a de T

$\delta(q1, ?, ?) = \{ (qf, \varepsilon) \}$



Exemplo: Autômato com Pilha Descendente

AP Descendente: Duplo Balanceamento

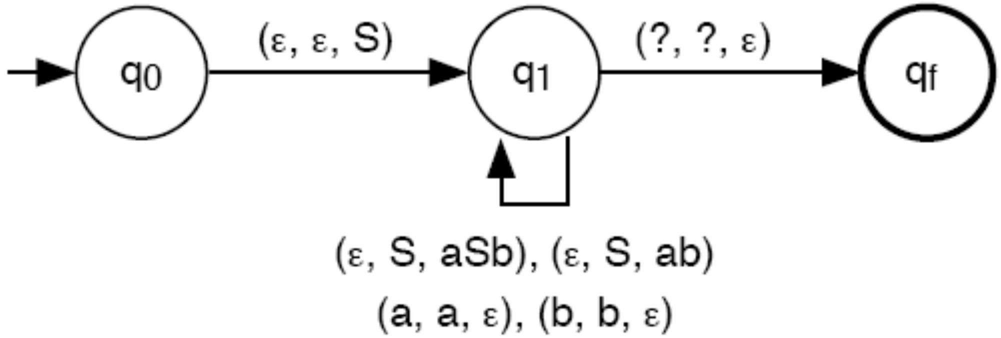
$L = \{ a^n b^n \mid n \geq 1 \}$

$G = (\{ S \}, \{ a, b \}, P, S)$ GLC sem recursão a esquerda

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

- Automato com pilha descendente

$M = (\{ a, b \}, \{ q0, q1, qf \}, \delta, q0, \{ qf \}, \{ S, a, b \})$



Reconhecedores de LLCs -

Análise sintática descendente:

– **Com retrocesso (back track):** Quando a gramática permite, em um determinado estágio da derivação, a aplicação de mais de uma regra. Isto acontece quando o mesmo símbolo terminal aparece no inicio do lado direito de mais de uma regra de produção.

Exemplo:

$A \rightarrow a\alpha$

$A \rightarrow a\beta$

onde: α , $\beta \in (V_N \cup V_T)^*$ $A \in V_N$ $a \in V_T$

Análise Sintática Descendente

Analise sintática descendente:

- Sem retrocesso (preditiva): Quando a gramática só permite um caminho a ser derivado. Desta forma, olhando apenas para o próximo símbolo de entrada podemos determinar a próxima derivação.

Requisitos:

- Durante o processo de derivação, sempre haverá uma única regra que possa levar a cadeia que esta sendo analisada.
- Não pode haver recursão a esquerda.

Ex : A -> A α onde $\alpha \in (V_N \cup V_T)^*$ e $A \in V_N$

Parser Preditivo Recursivo
ou Analisador Preditivo Recursivo

É executado um conjunto de procedimentos recursivos para processar a entrada. A cada não terminal é associado um procedimento;

- Existe também um procedimento adicional (Analisador Léxico) para o reconhecimento dos simbolos (tokens) ;

Vantagens:

- Simplicidade;

Desvantages:

- Maior tempo de processamento;
- Não é geral, pois existem linguagens que não aceitam recursividade.

Parser Preditivo Recursivo ou Analisador Preditivo Recursivo

- First(A) (ou Primeiro(A)) é o conjunto de tokens (símbolos) que figuram como primeiro elemento de uma ou mais cadeias geradas a partir de A.

Ex: S -> AS | BA

A -> aB | C

B -> bA | d

C -> c

First(S) = First(A) U First(B) = {a, c, b, d}

First(A) = {a, c}

First(B) = {b, d}

First(C) = {c}

Parser Preditivo Recursivo ou Analisador Preditivo Recursivo

Só se pode aplicar o analisador preditivo recursivo em uma gramática se para todas as regras do tipo A -> α, A -> β, ...(onde α , β ϵ $(V_N \cup V_T)^*$)os conjuntos First(α) e First(β) forem disjuntos

Algoritmo de Early

possivelmente o mais rápido algoritmo para LLC em geral

tempo de processamento proporcional a:

– em geral: $|w|^3$

– gramáticas nao-ambiguas: $|w|^2$

– muitas gramáticas de interesse prático: $|w|$

Algoritmo top-down

– a partir de uma GLC sem produções vazias

– parte do simbolo inicial

– executa sempre a derivação mais a esquerda

– cada ciclo gera um terminal

● compara com o símbolo da entrada

● sucesso -> construção do conjunto de produções que, potencialmente, pode gerar o próximo símbolo

Algoritmo de Early

Seja G = (V, T, P, S) uma GLC sem produções vazias

– $w = a_1a_2\ldots a_n$ palavra a ser verificada

- marcador "•”

– antecedendo a posição, em cada produção, que será analisada na tentativa de gerar o próximo simbolo terminal

- sufixo "/u" adicionado a cada produção

– indica o u-ésimo ciclo em que passou a ser considerada

Algoritmo de Early

Etapa 1: construção de D_0: primeiro conjunto de produções

(1)produções que partem de S

(2)produções que podem ser aplicadas em sucessivas derivações mais a esquerda (a partir de S)

- $D0 = \emptyset$
- para toda $S \rightarrow \alpha \in P$ (1)
- faca $D_0 = D_0 \cup \{ S \rightarrow \bullet\alpha/0 \}$
- repita para toda $A \rightarrow \bullet B\beta/0 \in D_0$ (2)

– faca para toda $B \rightarrow \varphi \in P$

- faca $D_0 = D_0 \cup \{ B \rightarrow \bullet\varphi/0 \}$

até que D_0 nao aumente

Algoritmo de Early

Etapa 2: construção dos demais conjuntos de produção

– • n = |w| conjuntos de produção a partir de D_0

– • ao gerar a_r, constroi D_r: produções que podem gerar a_{r+1}

● para r variando de 1 ate n (1)

● faça $D_r = \emptyset$;

para toda $A \rightarrow \alpha \bullet a_r \beta / s \in D_{r-1}$ (2)

– faça $D_r = D_r \cup \{ A \rightarrow \alpha a_r \bullet \beta / s \}$;

– repita para toda $A \rightarrow \alpha \bullet B \beta / s \in D_r$ (3)

● faça para toda $B \rightarrow \varphi \in P$ faça $D_r = D_r \cup \{ B \rightarrow \bullet \varphi / r \}$

● para toda $A \rightarrow \alpha \bullet / s$ de D_r (4)

● faça para toda $B \rightarrow \beta \bullet A \varphi / k \in D_s$ faça $D_r = D_r \cup \{ B \rightarrow \beta A \bullet \varphi / k \}$

– até que D_r não aumente

Algoritmo de Early

(1) cada ciclo gera um conjunto de produções D_r

(2) gera o simbolo a_r

(3) produções que podem derivar o próximo simbolo

(4) uma subpalavra de w foi reduzida a variável A

– inclui em D_r todas as produções de D_s que referenciaram •A;

Algoritmo de Early

Etapa 3: condição de aceitação da entrada.

– uma produção da forma $S \rightarrow \alpha\bullet/0$ pertence a D_n

w foi aceita se

$S \rightarrow \alpha\bullet/0$ e uma produção que

- parte do simbolo inicial S
- foi incluída em D_0 ("/0")
- todo o lado direito da produção foi analisado com sucesso ("•" esta no final de α)

Otimização do Algoritmo de Early

– ciclos repita-ate podem ser restritos exclusivamente as produções recentemente incluidas em D_r ou em D_0 ainda não-analisadas.

Algoritmo de Early

"Expressao simples" da linguagem Pascal

$G = (\{ E, T, F \}, \{ +, *, [,], x \}, P, E)$, na qual:

$P = \{ E \rightarrow T \mid E+T,\ T \rightarrow F \mid T*F,\ F \rightarrow [E] \mid x \}$

- Reconhecimento da palavra $x*x$
- D_0:

– $E \rightarrow \bullet T/0$	produções que partem
– $E \rightarrow \bullet E+T/0$	do símbolo inicial
– $T \rightarrow \bullet F/0$	produções que podem ser aplicadas
– $T \rightarrow \bullet T*F/0$	em derivação mais a esquerda
– $F \rightarrow \bullet [E]/0$	a partir do símbolo inicial
– $F \rightarrow \bullet x/0$	

Algoritmo de Early

D_1: reconhecimento de x em $x*x$

– $F \rightarrow x\bullet/0$ x	foi reduzido a F
– $T \rightarrow F\bullet/0$	inclui todas as produções de D_0 que
– $T \rightarrow T\bullet*F/0$	referenciaram •F direta ou indiretamente
– $E \rightarrow T\bullet/0$	movendo o marcador "•"
– $E \rightarrow E\bullet+T/0$	um simbolo para a direita

D_2: reconhecimento de $*$ em $x*x$

– $T \rightarrow T*\bullet F/0$	gerou $*$; o proximo sera gerado por F
– $F \rightarrow \bullet[E]/2$	inclui todas as producoes de P que
– $F \rightarrow \bullet x/2$	podem gerar o prox terminal a partir de F•

Algoritmo de Early

D_3: reconhecimento de x em $x * x$

– $F \rightarrow x\bullet/2$	x foi reduzido a F
– $T \rightarrow T*F\bullet/0$	incluido de D_2 (pois $F \rightarrow x\bullet/2$);
	entrada reduzida a T
– $E \rightarrow T\bullet/0$	incluido de D_0 (pois $T \rightarrow T*F\bullet/0$);
	entrada reduzida a E
– $T \rightarrow T\bullet*F/0$	incluido de D_0 (pois $T \rightarrow T*F\bullet/0$)
– $E \rightarrow E\bullet+T/0$	incluido de D_0 (pois $E \rightarrow T\bullet/0$)

- Como $w = x*x$ foi reduzida a E e
  como $E \rightarrow T\bullet/0$ pertence a D_3

Então a entrada é aceita

Algoritmo de Cocke-Younger-Kasami

a partir de uma GLC na Forma Normal de Chomsky

– gera bottom-up todas as arvores de derivacao da entrada w

– tempo de processamento proporcional a $|w|^3$

- idéia basica

– tabela triangular de derivação

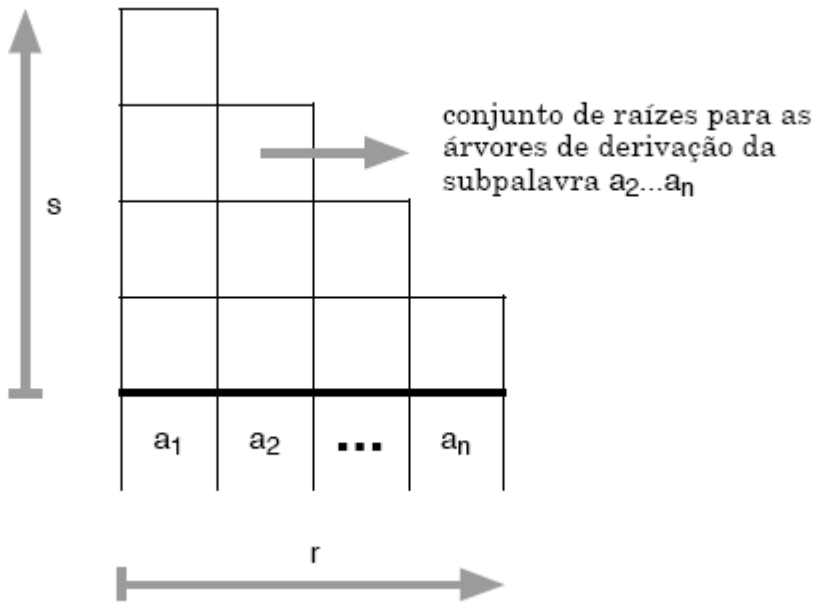
– célula: raizes que podem gerar a correspondente sub-arvore

Algoritmo de Cocke-Younger-Kasami

Seja G = (V, T, P, S) uma GLC na Forma Normal de Chomsky

$w = a_1a_2\ldots a_n$ uma entrada entrada

V_{rs} células da tabela



Algoritmo de Cocke-Younger-Kasami

Etapa 1: variáveis que geram diretamente terminais ($A \rightarrow a$)

– para r variando de 1 ate n faca $V_{r1} = \{ A \mid A \rightarrow a_r \in P \}$

Etapa 2: produções que geram duas variaveis ($A \rightarrow BC$)

– para s variando de 2 ate n

– faça para r variando de 1 ate n - s + 1

faça $V_{rs} = \emptyset$

– para k variando de 1 ate s - 1

– faça $V_{rs} = V_{rs} \cup \{ A \mid A \rightarrow BC \in P,\ B \in V_{rk} \text{ e } C \in V_{(r+k)(s-k)} \}$

- limite de iteração para r e (n - s + 1): a tabela é triangular
- V_{rk} e $V_{(r+k)(s-k)}$ são as raizes das sub-árvores de V_{rs}
- celula vazia: não gera qualquer sub-árvore

Etapa 3: condição de aceitação da entrada.

símbolo inicial pertence a V_{1n} (raiz de toda palavra)

Algoritmo de Cocke-Younger-Kasami

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

P = { $S \rightarrow AA \mid AS \mid b$, $A \rightarrow SA \mid AS \mid a$ }

S,A				
S,A	S,A			
S,A	S	S,A		
S,A	A	S	S,A	
A	S	A	A	S
a	b	a	a	b

Como S é raiz da árvore de derivação a entrada é aceita

Algoritmo de Cocke-Younger-Kasami

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

P = { S → AA | AS | b, A → SA | AS | a }

1. $s = 2$, $r = 1$, $k = 1$: $A \to BC$, $B \in V(1,1)$ e $C \in V(2,1)$. $V(1,2) = \emptyset \bigcup \{S, A\} = \{S, A\}$;
2. $s = 2$, $r = 2$, $k = 1$: $B \in V(2,1)$ e $C \in V(3,1)$. $V(2,2) = \emptyset \bigcup \{A\} = \{A\}$;
3. $s = 2$, $r = 3$, $k = 1$: $B \in V(3,1)$ e $C \in V(4,1)$. $V(3,2) = \emptyset \bigcup \{S\} = \{S\}$;
4. $s = 2$, $r = 4$, $k = 1$: $B \in V(4,1)$ e $C \in V(5,1)$. $V(4,2) = \emptyset \bigcup \{S, A\} = \{S, A\}$;
5. $s = 3$, $r = 1$, $k = 1$: $B \in V(1,1)$ e $C \in V(2,2)$. $V(1,3) = \emptyset \bigcup \{S\} = \{S\}$;

Algoritmo de Cocke-Younger-Kasami

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

P = { S → AA | AS | b, A → SA | AS | a }

6. $s = 3$, $r = 1$, $k = 2$: $B \in V(1,2)$ e $C \in V(3,1)$.
   $V(1,3) = \{S\} \bigcup \{S, A\} = \{S, A\}$;
7. $s = 3$, $r = 2$, $k = 1$: $B \in V(2,1)$ e $C \in V(3,2)$.
   $V(2,3) = \emptyset \bigcup \emptyset = \emptyset$;
8. $s = 3$, $r = 2$, $k = 2$: $B \in V(2,2)$ e $C \in V(4,1)$.
   $V(2,3) = \emptyset \bigcup \{S\} = \{S\}$;
9. $s = 3$, $r = 3$, $k = 1$: $B \in V(3,1)$ e $C \in V(4,2)$.
   $V(3,3) = \emptyset \bigcup \{S, A\} = \{S, A\}$;
10. $s = 3$, $r = 3$, $k = 2$: não precisa.
11. $s = 4$, $r = 1$, $k = 1$: $B \in V(1,1)$ e $C \in V(2,3)$.
   $V(1,4) = \emptyset \bigcup \{S, A\} = \{S, A\}$;

Algoritmo de Cocke-Younger-Kasami

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

P = { S → AA | AS | b, A → SA | AS | a }

12. $s = 4$, $r = 1$, $k = 2, 3$: não precisa;
13. $s = 4$, $r = 2$, $k = 1$: $B \in V(1,1)$ e $C \in V(2,3)$.
    $V(2,4) = \emptyset \bigcup \{S, A\} = \{S, A\}$;
14. $s = 4$, $r = 2$, $k = 2, 3$: não precisa;
15. $s = 5$, $r = 1$, $k = 1$: $B \in V(1,1)$ e $C \in V(2,4)$.
    $V(1,5) = \emptyset \bigcup \{S, A\} = \{S, A\}$;
16. $s = 5$, $r = 1$, $k = 2, 3, 4$: não precisa.

Hierarquia de Chomsky

Hierarquia de Chomsky

- Descreve as classes de linguagens formais

Hierarquia	Gramáticas	Linguagens	Autômato mínimo
Tipo-0	?	?	?
Tipo-1	?	?	?
Tipo-2	Livres de contexto	Livres de contexto	Autômatos de pilha
Tipo-3	Regulares (Expressões regulares)	Regulares	Autômatos finitos

Hierarquia de Chomsky

- Podemos detalhar o tipo-2

Hierarquia	Gramáticas	Linguagens	Autômato mínimo
Tipo-0	?	?	?
Tipo-1	?	?	?
Tipo-2	Livres de contexto	Livres de contexto	Autômatos de pilha não-determinísticos (NPDA)
	Livres de contexto determinísticas	Livres de contexto determinísticas	Autômatos de pilha determinísticos (DPDA)
Tipo-3	Regulares (Expressões regulares)	Regulares	Autômatos finitos (NFA, DFA, ε-NFA

Hierarquia de Chomsky

- Podemos detalhar ainda mais a classe das gramáticas/linguagens determinísticas

<table>
<tr><th>Hierarquia</th><th colspan="4">Gramáticas</th><th colspan="4">Linguagens</th><th colspan="4">Autômato mínimo</th></tr>
<tr><td>Tipo-0</td><td colspan="4">?</td><td colspan="4">?</td><td colspan="4">?</td></tr>
<tr><td>Tipo-1</td><td colspan="4">?</td><td colspan="4">?</td><td colspan="4">?</td></tr>
<tr><td>Tipo-2</td><td colspan="4">Livres de contexto</td><td colspan="4">Livres de contexto</td><td colspan="4">Autômatos de pilha não-determinísticos (NPDA)</td></tr>
<tr><td></td><td>LL</td><td>LR</td><td>LALR</td><td>...</td><td>LL</td><td>LR</td><td>LALR</td><td>...</td><td>LL</td><td>LR</td><td>LALR</td><td>...</td></tr>
<tr><td>Tipo-3</td><td colspan="4">Regulares
(Expressões regulares)</td><td colspan="4">Regulares</td><td colspan="4">Autômatos finitos (NFA, DFA, ε-NFA</td></tr>
</table>

Fim