Disciplina de Linguagens Formais

Prof. Carlos A. P. Campani

11 de abril de 2005

Objetivos

- Formalizar o conceito de linguagem;
- Apresentar uma classificação de linguagens baseada nesta formalização;
- Fornecer ferramentas para implementar compiladores;
- Relacionar a hierarquia de linguagens com o conceito de computabilidade.

Súmula da Disciplina

1. Introdução à Teoria de Linguagens Formais;
   - Alfabetos, Sentenças e Linguagens;
   - Gramáticas e Reconhecedores;
   - Tipos de Gramáticas/Linguagens (Hierarquia de Chomsky);

2. Gramáticas Regulares e Autômatos Finitos;
   - Definição de Gramática Regular (GR);
   - Definição de Autômato Finito;
   - Autômato Finito Não-Determinístico;
   - Relação entre Autômato Finito e GR;
   - Minimização de Autômato Finito;
   - Expressões Regulares;
   - Bombeamento para Linguagens Regulares;
   - Máquinas de Moore e Mealy;

3. Gramáticas Livres de Contexto e Autômatos de Pilha;
   - Definição de GLC;
   - Árvores de Derivação e Ambigüidade;
   - Transformações em GLC: Eliminação de Símbolos Inúteis; Transformação em uma GLC ε-Livre; Remoção de Produções Simples; Fatoração; Eliminação de Recursão à Esquerda;
   - Formas Canônicas: Forma Normal de Chomsky; Forma Normal de Greibach;
   - Autômatos de Pilha (PDA);
   - Relação entre PDA e GLC;

4. Reconhecedor – Algoritmo CYK;
5. Linguagens Tipo 0 e Máquinas de Turing;
6. Linguagens Sensíveis ao Contexto e Autômatos de Fita Limitada;
7. Hierarquia de Chomsky (revisitada).

Bibliografia

- *Introduction to Automata Theory, Languages, and Computation*, Hopcroft e Ullman;
- *Linguagens Formais e Autômatos*, Paulo Blauth Menezes.

Links

(Lâminas)

`http://www.ufpel.tche.br/~campani/lingformal.pdf`

(Lâminas para impressão)

`http://www.ufpel.tche.br/~campani/lingformal4.ps.gz`

1 Introdução

- Uma *linguagem* é um meio de comunicação utilizado por elementos de uma determinada comunidade. Formada por:
  - Conjunto de *palavras*;
  - Conjunto de *regras gramaticais*;
- Uma linguagem é *formal* quando pode ser representada através de um sistema com sustentação matemática. Formada por:
  - Sintaxe (estrutura) – foco da nossa disciplina;
  - Semântica (significado) – não interessa para nós.

- *Alfabeto* (ou *vocabulário*) é um conjunto finito de símbolos. Ex: dígitos, letras, etc.

$$\{a, b, c, \ldots, z\}$$

$$\{0, 1\}$$

- *Sentença* (ou *palavra*, ou *string*) sobre um alfabeto é qualquer seqüência finita de símbolos do alfabeto. Ex: Seja o alfabeto $V = \{0, 1\}$, são sentenças válidas 001, 1010, etc.

- O *tamanho* (ou *comprimento*) de uma sentença é dado pelo número de símbolos que a compõe. Ex: $V = \{a, b, c\}$, sentença $w = ab$, tamanho $|w| = 2$;

- A sentença que não contém símbolos (tamanho zero) é chamada de *sentença vazia*, sendo denotada por ε ($|\varepsilon| = 0$);
- Seja V um alfabeto. Representamos por:
  - V^* – conjunto de todas as sentenças compostas de símbolos de V incluindo a sentença vazia;
  - $V^+ = V^* - \{\varepsilon\}$;

  Exemplo: $V = \{0, 1\}$,
  $V^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$,
  $V^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$.

Linguagem é qualquer conjunto de sentenças sobre um alfabeto.

ou

Uma *linguagem* L sobre um alfabeto V é tal que $L \subset V^*$.

- Tipos de linguagens: finitas, vazias e infinitas;
- Representação:
  - Listando as palavras (só finita);
  - Descrição algébrica. Ex: $\{a^n b^n c^n | n \geq 1\}$.
    Notação:
    $$\overbrace{aaa\cdots a}^{n \text{ vezes}} = a^n ;$$
  - Definindo um algoritmo que determina (sim ou não) se uma sentença pertence a uma linguagem (*Reconhecimento*). Ex: autômato finito;
  - Definindo um mecanismo que gera todas as sentenças da linguagem em alguma ordem (*Geração*). Ex: gramática.

- Uma *gramática* serve para definir qual o subconjunto de sentenças que faz parte de uma determinada linguagem. Ela é um dispositivo formal para especificar uma linguagem potencialmente infinita de uma forma finita;

- Notação Formal de Gramática

  Uma gramática G é definida por uma quádrupla

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

  onde:

  N conjunto finito de não-terminais;

  T conjunto finito de terminais;

  P conjunto finito de regras de produção;

  S símbolo inicial da gramática.

  Observações: $N \bigcap T = \emptyset$, $V = N \bigcup T$, $S \in N$,
  $P = \{\alpha \rightarrow \beta | \alpha \in V^+ \wedge \beta \in V^*\}$.

Exemplo: gera números inteiros

$$G = (N, T, P, I)$$

$$N = \{I, D\}$$

$$T = \{0, 1, 2, \ldots, 9\}$$

$$P = \{I \to D, I \to DI, D \to 0, D \to 1, \ldots, D \to 9\}$$

ou

$$P : I \to D|DI, D \to 0|1|2|\cdots|9$$

Exercício: Reescrever tentando evitar números tipo 0015.

- Convenções:

  Símbolos não-terminais $N = \{A, B, C, \ldots T\}$;

  Símbolos terminais $T = \{a, b, c, \ldots t\}$;

  Símbolo terminal ou não-terminal

  $N \bigcup T = \{U, V, \ldots Z\}$;

  Seqüências de terminais $T^* = \{u, v, \ldots z\}$;

  Seqüências de não-terminais e terminais

  $(N \bigcup T)^* = \{\alpha, \beta, \gamma, \ldots\}$.

- *Derivação* ($\Rightarrow$) é uma operação de substituição efetuada de acordo com as regras de produção da gramática.

  Formalmente: $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ se $\alpha \rightarrow \beta \in P$ e $\gamma, \delta \in V^*$.

  Exemplo: $G = (\{S, A, B\}, \{0, 1\}, \{S \rightarrow AB, A \rightarrow 0|AB, B \rightarrow 1|1B\}, S)$, e

  $$S \Rightarrow AB \Rightarrow ABB \Rightarrow 0BB \Rightarrow 01B \Rightarrow 011B\,.$$

- Uma seqüência de derivações com zero ou mais passos ($\Rightarrow^*$):

  Formalmente: $\alpha_1 \Rightarrow^* \alpha_m$ se
  $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha 3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$,
  $\alpha_1, \alpha_2, \ldots, \alpha_m \in V^*$.

- Garantindo pelo menos uma substituição ($\Rightarrow^+$):

  Formalmente: $\alpha_1 \Rightarrow^+ \alpha_m$ se $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow^* \alpha_m$,
  $\alpha_1, \alpha_2, \ldots, \alpha_m \in V^*$.

- Uma *sentença* de uma linguagem é uma seqüência formada por terminais, obtida a partir do símbolo inicial da gramática desta linguagem, através de derivações sucessivas.

  Formalmente: $S \Rightarrow^{+} u$.

- Uma *forma sentencial* é uma seqüência qualquer, composta de terminais e não-terminais, obtida a partir do símbolo inicial da gramática através de derivações sucessivas.

  Formalmente: $S \Rightarrow^{*} \alpha$.

- A linguagem gerada por uma gramática $G = (N, T, P, S)$ é o conjunto de todas as sentenças da linguagem obtidas da gramática por derivações sucessivas.

  Formalmente: $L(G) = \{w | w \in T^* \wedge S \Rightarrow^* w\}$.

Ex: $G = (\{S\}, \{0,1\}, P, S)$ com
$P = \{S \to 0S1, S \to 01\}$. Assim,
$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0^3S1^3 \Rightarrow \cdots$ que é igual a 0^n1^n
para $n \geq 1$. Logo, $L(G) = \{0^n1^n | n \geq 1\}$.

Notação:

$$\overbrace{aaa\cdots a}^{n \text{ vezes}} = a^n \, .$$

Exercício: Seja a seguinte gramática:

$$G = (\{S, A, B\}, \{0, 1\}, P, S)$$

$$P = \{S \rightarrow ASBB|BSAA|0, A \rightarrow 01|01A, B \rightarrow 10|10B\}$$

Obtenha as derivações para as seguintes sentenças:

1. 0101010;
2. 101000101010101.

- Tipos de Gramáticas – segundo a hierarquia de Chomsky existem quatro tipos:

  Tipo 0 ou Irrestritas $P = \{\alpha \rightarrow \beta | \alpha \in V^+, \beta \in V^*\}$;

  Tipo 1 ou Sensível ao Contexto Se $\alpha \rightarrow \beta \in P$ então $|\alpha| \leq |\beta|$.

  Outra forma:

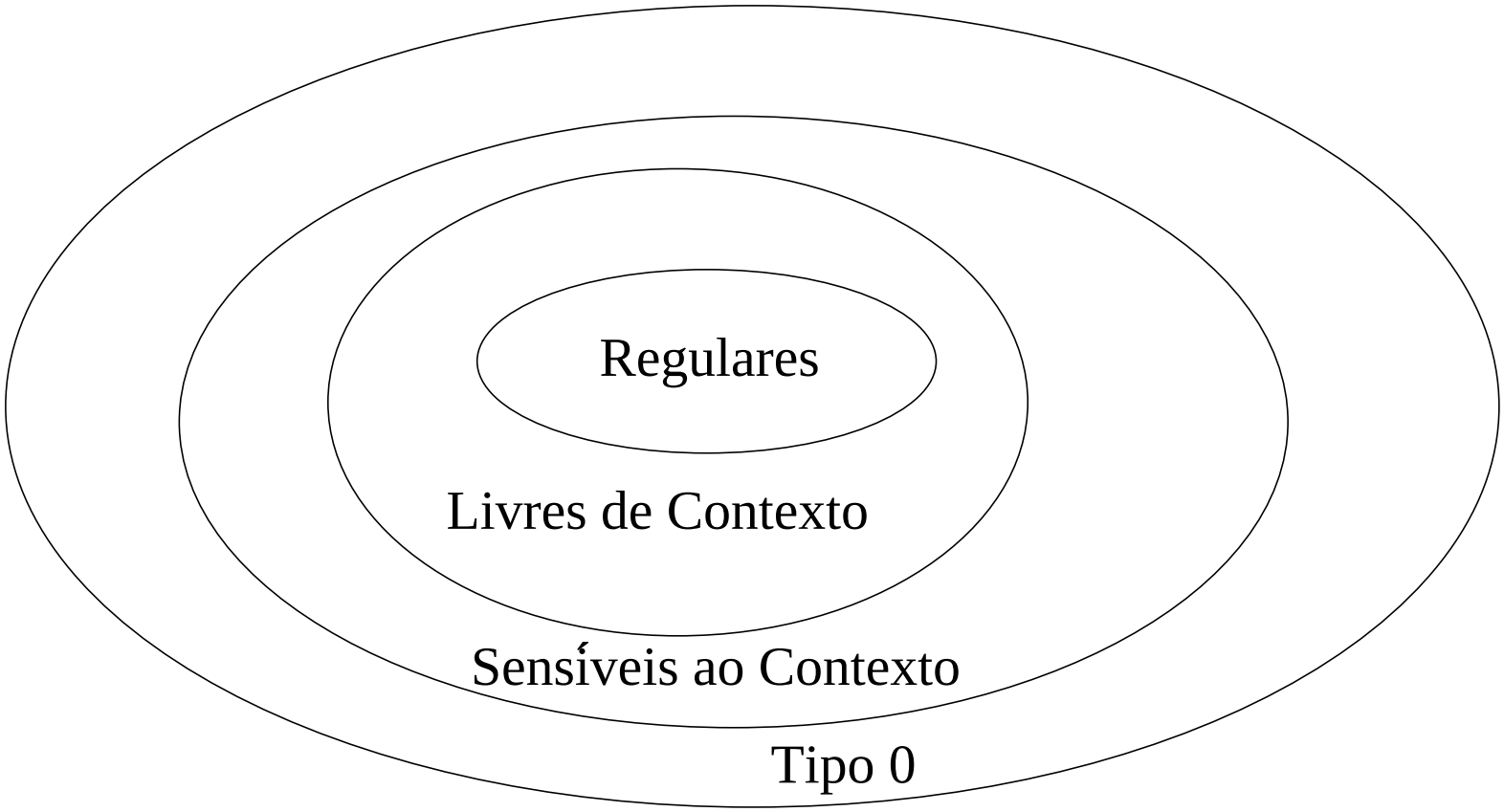
  $P = \{\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 | \alpha_1, \alpha_2 \in V^*, \beta \in V^+, A \in N\}$;

  Tipo 2 ou Livre de Contexto

  $P = \{\alpha \rightarrow \beta | \alpha \in N \wedge \beta \neq \varepsilon\}$;

  Tipo 3 ou Regular $A \rightarrow aB$ ou $A \rightarrow a$, ou seja,

  $P = \{A \rightarrow aX | A \in N, a \in T, X \in N \cup \{\varepsilon\}\}$.



$$\text{Tipo } 3 \subset \text{Tipo } 2 \subset \text{Tipo } 1 \subset \text{Tipo } 0 \subset T^*$$

- Ex:

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

$$P = \{S \to aB|bA, A \to a|aS|bAA, B \to b|bS|aBB\}$$

1. Tipo da gramática?
2. $L(G)$?

- Respostas:
  1. Livre de contexto;
  2. $L(G)$ é o conjunto de todas as palavras em T^* que tem o mesmo número de a's e b's.

- Linguagens classificadas segundo as gramáticas em:
  1. Recursivamente enumeráveis ou RE (gramática Tipo 0) – conjuntos recursivamente enumeráveis estão relacionado com a máquina de Turing, funções parciais recursivas e computabilidade;
  2. Linguagem sensível ao contexto ou LSC (gramática Tipo 1);
  3. Linguagem livre de contexto ou LLC (gramática Tipo 2);
  4. Linguagem Regular ou LR (gramática Tipo 3).

- Vamos extender as definições das gramáticas Tipo 1 e 2 para permitir produções do tipo $S \rightarrow \varepsilon$. Isto só é possível *se S não aparece em lado direito de nenhuma produção*. Assim, $S \rightarrow \varepsilon$ somente será utilizada para originar a sentença vazia;

$$L(G') = L(G) \bigcup \{\varepsilon\}$$

- Para introduzir $S \rightarrow \varepsilon$ em gramática que viola a condição anterior, devemos transforma-la em uma *equivalente* que obedeça a restrição;
- *Gramáticas equivalentes* são aquelas que geram a mesma linguagem;

- Para transformar incluiremos um novo símbolo não-terminal (S') que passará a ser o novo símbolo inicial. Em seguida, incluiremos em P todas as regras que tenham o símbolo S no lado esquerdo, substituindo este por S', e incluindo ainda uma regra $S' \rightarrow \varepsilon$;

  Ex1: $G = (\{S, A\}, \{a\}, P, S)$ com
  $P = \{S \rightarrow aA, A \rightarrow a|aA\}$. Obtemos
  $G' = (\{S, A\}, \{a\}, P', S)$ com
  $P' = \{S \rightarrow aA|\varepsilon, A \rightarrow a|aA\}$;

Ex2: $G = (\{S, A\}, \{a\}, P, S)$ com
$P = \{S \to aA, A \to a|aS\}$. Obtemos
$G' = (\{S', S, A\}, \{a\}, P', S')$ com
$P' = \{S' \to aA|\varepsilon, S \to aA, A \to a|aS\}$;

Exercícios:

1. Construa uma gramática G tal que:
   (a) $L(G) = \{a^n b^m | n \geq 0 \wedge m \geq 1\}$;
   (b) $L(G) = \{a^i b^j c^i | i \geq 0 \wedge j \geq 1\}$.
2. Construa uma gramática regular G tal que:

   $L(G) = \{w | w \in \{0,1\}^+$ e não tem $1's$ consecutivos$\}$
3. Construa uma gramática livre de contexto G tal que:

   $L(G) = \{w | w \in$
   $\{0,1,2\}^+$ e todos os zeros sejam consecutivos$\}$

4. Seja G definida por:

$$S \rightarrow aS|bB, B \rightarrow cB|cC, C \rightarrow cS|c$$

Pede-se:

(a) determine $L(G)$;

(b) construa G' tal que $L(G') = L(G) \bigcup \{\varepsilon\}$;

(c) verifique se as sentenças abaixo pertencem a $L(G)$:

- $abccc$;
- $bccabcc$;
- $abccbbc$.

5. Construa uma GLC G tal que $L(G)$ seja o conjunto de expressões aritméticas válidas. Os terminais são identificadores (id), parênteses ("(" e ")"), e os operadores $+$, $*$, e $-$ unário.

2 Gramáticas Regulares e Autômato Finito

- O *autômato finito* (AF) é uma máquina abstrata que reconhece *linguagens regulares*;
- Modelo matemático com entradas e saídas. Se é fornecido ao AF uma seqüência de símbolos como entrada, ele responderá se esta seqüência pertence à linguagem ou não;
- O AF pode assumir um número finito de estados;
- Um estado resume os estados anteriores pelos quais passou e os símbolos que já foram lidos na entrada.

- Definição formal de autômato finito

  Um *autômato finito* M sobre um alfabeto Σ é um sistema $(K, \Sigma, \delta, q_0, F)$ onde:

  K – conjunto finito, não vazio, de estados;

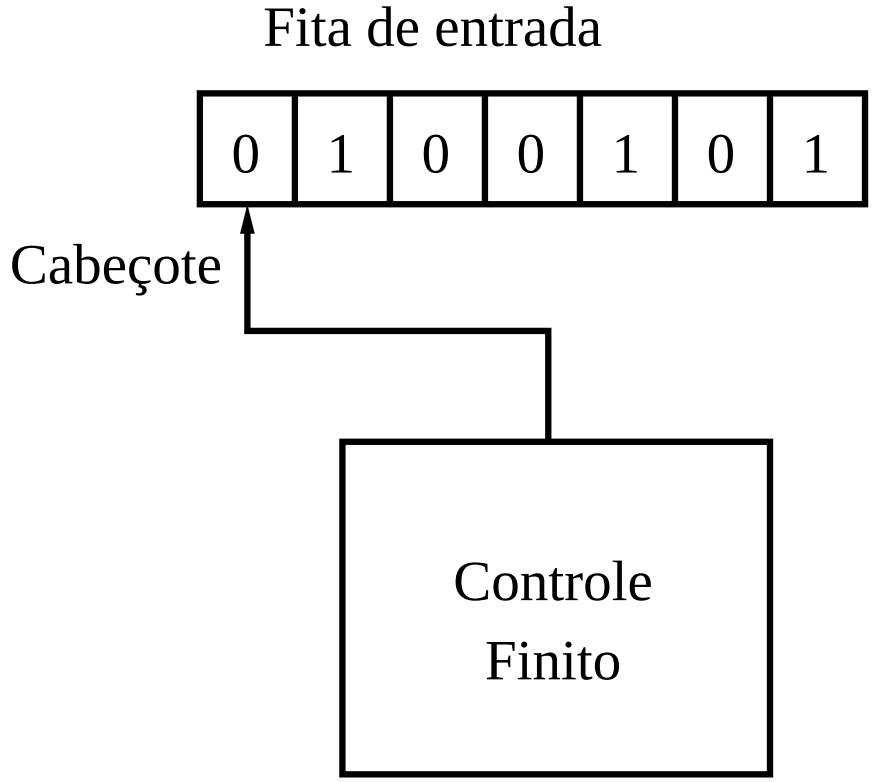
  Σ – alfabeto finito de entrada;

  δ – função de transição de estados,

  $$\delta : K \times \Sigma \bigcup \{\varepsilon\} \rightarrow K\,;$$

  $q_0 \in K$ estado inicial;

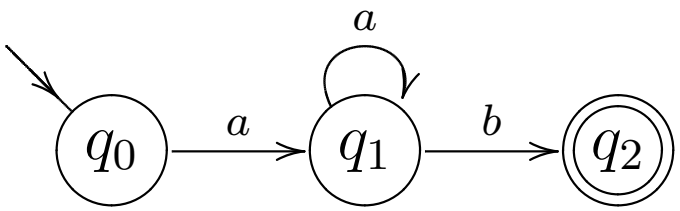
  $F \subset K$ conjunto de estados finais.



$\delta(q, a) = p$ para $q, p \in K$ e $a \in \Sigma$, significando que estando no estado q e lendo o símbolo a na fita de entrada podemos mudar para o estado p avançando o cabeçote uma posição.

- Diagrama de Transição
  - Outra maneira de representar um autômato finito;
  - Grafo direcionado e rotulado;
  - Os vértices representam os estados, desenhados como círculos;
  - As arestas representam as transições entre dois estados, sendo o rótulo o símbolo reconhecido na entrada;
  - O estado inicial é marcado com uma seta;
  - Os estados finais são indicados por círculos duplos.

Ex:



$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$, e $\delta(q_0, a) = q_1$,
$\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_2$. Logo, $T(M) = \{a^n b | n \geq 1\}$.

- Tabela de Transição de Estados
  - Uma terceira forma de representar um AF;
  - Tabela indicando na vertical os estados, e na horizontal os símbolos do alfabeto. No cruzamento estão as transições possíveis;
  - O estado inicial é indicado por uma seta, e os estados finais por asteriscos.

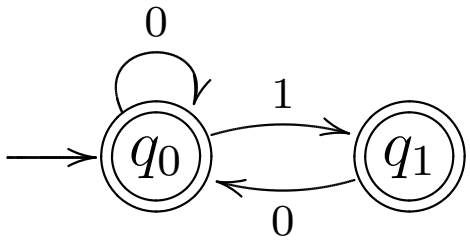
Ex1:

δ	a	b
$\longrightarrow q_0$	q_1	-
q_1	q_1	q_2
$^*q_2$	-	-

Exercício:

1. Descreva formalmente o AF;
2. Desenhe o diagrama de transição equivalente.

Ex2: AF que reconhece sentenças em $\{0,1\}^*$ as quais não contém dois ou mais 1's consecutivos.



$M = (K, \Sigma, \delta, q_0, F)$, $K = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$,
$F = \{q_0, q_1\}$, com $\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$, $\delta(q_1, 0) = q_0$.

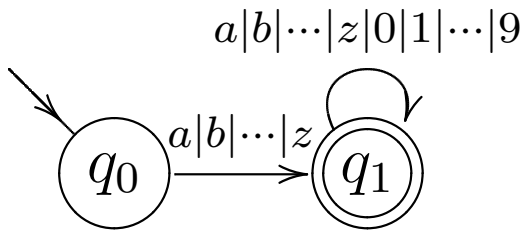
Exercício: Apresente a tabela de transição.

Tabela:

δ	0	1
$\longrightarrow *q_0$	q_0	q_1
$^*q_1$	q_0	-

Ex3: Identificadores de linguagens de programação

$S \to L|LR, R \to L|D|LR|DR, L \to a|b|c\cdots|z, D \to 0|1|2|\cdots|9$



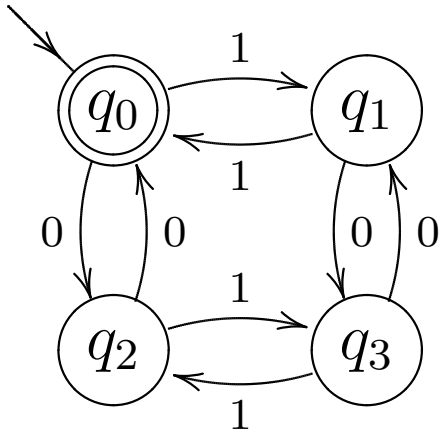
Exercícios:

1. Inteiros com ou sem sinal;
2. Reais (uma casa após a vírgula) com sinal opcional.

- Extensão da função δ para o domínio $K \times \Sigma^*$:
  1. $\hat{\delta}(q, \varepsilon) = q$;
  2. $\hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a)$ para $u \in \Sigma^*$ e $a \in \Sigma$;
- $\hat{\delta}(q, u) = p$ significa que M, iniciando no estado q e tendo a seqüência u na fita de entrada, entrará no estado p, após o cabeçote mover-se para a direita $|u|$ células;
- Uma sentença u é aceita por M se $\hat{\delta}(q_0, u) = p$ para algum $p \in F$, ou seja:
  $T(M) = \{u | \hat{\delta}(q_0, u) = p \wedge p \in F\}$;
- Qualquer conjunto de sentenças aceito por um AF é dito ser *regular* (*linguagem regular*).

Exemplo: $M = (K, \Sigma, \delta, q_0, F)$, $\Sigma = \{0, 1\}$,
$K = \{q_0, q_1, q_2, q_3\}$, $F = \{q_0\}$.

$$
\begin{aligned}
\delta(q_0, 0) &= q_2 \\
\delta(q_1, 0) &= q_3 \\
\delta(q_2, 0) &= q_0 \\
\delta(q_3, 0) &= q_1 \\
\delta(q_0, 1) &= q_1 \\
\delta(q_1, 1) &= q_0 \\
\delta(q_2, 1) &= q_3 \\
\delta(q_3, 1) &= q_2
\end{aligned}
$$


q_0
q_1
q_2
q_3
1
1
0
0
0
0
1
1

δ	0	1
$\longrightarrow *q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Reconhecimento de 110101:

$(q_0, 110101) \Rightarrow (q_1, 10101) \Rightarrow (q_0, 0101) \Rightarrow (q_2, 101) \Rightarrow (q_3, 01) \Rightarrow (q_1, 1) \Rightarrow (q_0, \varepsilon)$. Assim, $110101 \in T(M)$.

Reconhecimento de 1011:

$(q_0, 1011) \Rightarrow (q_1, 011) \Rightarrow (q_3, 11) \Rightarrow (q_2, 1) \Rightarrow (q_3, \varepsilon)$.

Assim, $1011 \notin T(M)$.

$T(M) = \{w | w \in$
$\{0,1\}^*$ e w contém um número par de $0'$s e $1'$s$\}$.

- Autômato Finito Não-Determinístico (AFND)

  Um *autômato finito não-determinístico* é um sistema $(K, \Sigma, \delta, q_0, F)$ onde:

  K conjunto finito, não vazio, de estados;

  Σ alfabeto de entrada;

  $\delta : K \times \Sigma \to \text{Partes}(K)$ função de transição (não determinístico);

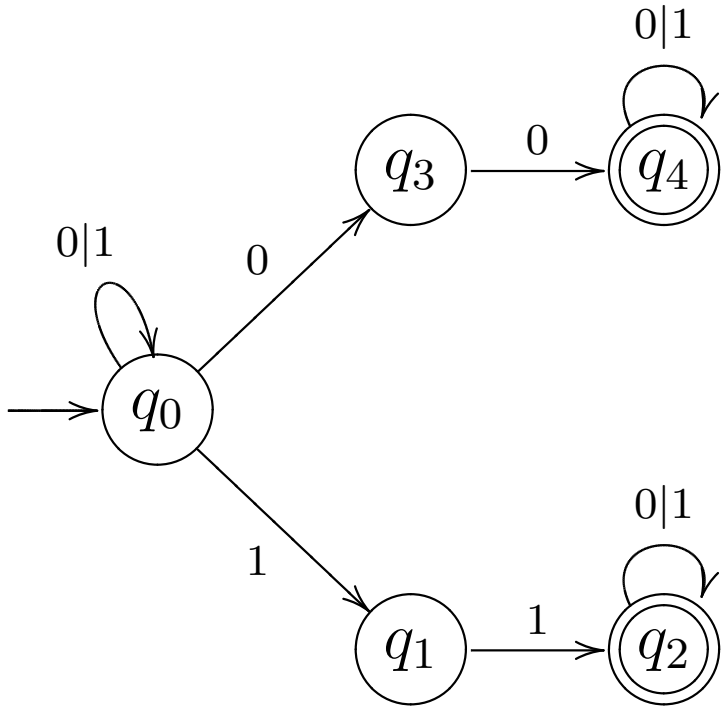
  $q_0 \in K$ estado inicial;

  $F \subset K$ conjunto de estados finais.

- Diferença entre AF e o AFND: $\delta(q, a)$ é um conjunto de estados (possivelmente vazio) ao invés de um único estado;
- Assim, $\delta(q, a) = \{p_1, p_2, \ldots, p_k\}$ significa que M estando no estado q e tendo a na posição do cabeçote na fita de entrada, move uma célula para a direita e escolhe qualquer estado entre $p_1, p_2, \ldots, p_k$ como próximo estado.

- Extendendo δ para o domínio $K \times \Sigma^*$:
  1. $\hat{\delta}(q, \varepsilon) = \{q\}$;
  2. $\hat{\delta}(q, ua) = \bigcup_{p \in \hat{\delta}(q,u)} \delta(p, a)$ para todo $u \in \Sigma^*$ e $a \in \Sigma$;
- Uma sentença u é aceita por M se existe um estado $p \in F$ e $p \in \hat{\delta}(q_0, u)$;
- $T(M) = \{u | p \in \hat{\delta}(q_0, u) \wedge p \in F\}$.

Ex: AFND que aceita o conjunto de todas as sentenças binárias que contém dois 0's ou 1's consecutivos.



$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_0, \{q_2, q_4\})$ e

$$
\begin{aligned}
\delta(q_0, 0) &= \{q_0, q_3\} \\
\delta(q_1, 0) &= \emptyset \\
\delta(q_2, 0) &= \{q_2\} \\
\delta(q_3, 0) &= \{q_4\} \\
\delta(q_4, 0) &= \{q_4\} \\
\delta(q_0, 1) &= \{q_0, q_1\} \\
\delta(q_1, 1) &= \{q_2\} \\
\delta(q_2, 1) &= \{q_2\} \\
\delta(q_3, 1) &= \emptyset \\
\delta(q_4, 1) &= \{q_4\}
\end{aligned}
$$

- Teorema: Se L é um conjunto aceito por um AFND, então existe um autômato finito determinístico (AFD) que aceita L.

  (Ou seja, o não determinismo não acrescenta nada – a não ser praticidade – ao AF).

- Seja $M = (K, \Sigma, \delta, q_0, F)$ um AFND que aceita L. Definimos o AFD $M' = (K', \Sigma, \delta', q_0', F')$ tal que:
  - Os estados de M' são constituidos por todos os subconjuntos do conjunto de estados de M: $K' = \text{Partes}(K)$ e $\overline{\overline{K'}} = 2^{\overline{\overline{K}}}$;
  - Notação: estados de M' são $[q_1 q_2 q_3 \cdots q_i]$, onde $q_1, q_2, q_3, \ldots, q_i \in K$;
  - Ex: Se $K = \{q_0, q_1\}$ então $K' = \{\emptyset, [q_0], [q_1], [q_0 q_1]\}$;
  - F' é o conjunto de todos os estados de K' contendo um estado de F;
  - $q_0' = [q_0]$;

– Definimos $\delta'([q_1q_2q_3\cdots q_i], a) = [p_1p_2p_3\cdots p_j]$ se e somente se $\delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\} = \delta(q_1, a) \cup \delta(q_2, a) \cup \cdots \cup \delta(q_i, a)$.

- Prova: $T(M) = T(M')$. Por indução sobre o tamanho da palavra de entrada w. Devemos mostrar que $\hat{\delta}'([q_0], w) = [p_1 \cdots p_i]$ se e somente se $\hat{\delta}(q_0, w) = \{p_1, \ldots, p_i\}$.
  1. Base: $|w| = 0$, $w = \varepsilon$. Então, $\hat{\delta}'([q_0], \varepsilon) = [q_0]$ se e somente se $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$, que é verdadeiro por definição;
  2. Hipótese: $|w| = n$, $n \geq 1$.
     Supor $\hat{\delta}'([q_0], w) = [p_1 \cdots p_i]$ se e somente se $\hat{\delta}(q_0, w) = \{p_1, \ldots, p_i\}$;

3. Passo: $|wa| = n + 1$, $n \geq 1$:

$$\hat{\delta}'([q_0], wa) = [q_1 \cdots q_j]$$

se e somente se

$$\hat{\delta}(q_0, wa) = \{q_1, \ldots, q_j\}$$

que equivale a dizer que

$$\hat{\delta}'([p_1 \cdots p_i], a) = [q_1 \cdots q_j]$$

se e somente se

$$\hat{\delta}(\{p_1, \ldots, p_i\}, a) = \{q_1, \ldots, q_j\}$$

que é verdade por definição.

Exemplo: Dado o AFND $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ e

$$\delta(q_0, 0) = \{q_0, q_1\} \quad \delta(q_0, 1) = \{q_1\}$$

$$\delta(q_0, 1) = \emptyset \quad \delta(q_1, 1) = \{q_0, q_1\}.$$

Construir o AFD M' equivalente.

$$M' = (K', \{0,1\}, \delta', [q_0], F')$$

$$K' = \{[q_0], [q_1], [q_0q_1], \emptyset\}$$

$$F' = \{[q_1], [q_0q_1]\}$$

$$\begin{aligned}
\delta(q_0, 0) = \{q_0, q_1\} &\Rightarrow \delta'([q_0], 0) = [q_0q_1] \\
\delta(q_0, 1) = \{q_1\} &\Rightarrow \delta'([q_0], 1) = [q_1] \\
\delta(q_1, 0) = \emptyset &\Rightarrow \delta'([q_1], 0) = \emptyset \\
\delta(q_1, 1) = \{q_0, q_1\} &\Rightarrow \delta'([q_1], 1) = [q_0q_1]
\end{aligned}$$

$$\begin{aligned}
\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \quad &\Rightarrow \quad \delta'([q_0 q_1], 0) = [q_0 q_1] \\
= \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\} & \\
\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \quad &\Rightarrow \quad \delta'([q_0 q_1], 1) = [q_0 q_1] \\
= \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\} &
\end{aligned}$$

Tabela do AFND:

δ	0	1
$\rightarrow q_0$	q_0, q_1	q_1
$*q_1$	$\emptyset$	q_0, q_1

Tabela do AFD equivalente:

δ'	0	1
$\rightarrow [q_0]$	$[q_0 q_1]$	$[q_1]$
$*[q_1]$	$\emptyset$	$[q_0 q_1]$
$*[q_0 q_1]$	$[q_0 q_1]$	$[q_0 q_1]$

- Relação entre AF e GR
- Teorema: Se $G = (N, T, P, S)$ é uma GR (Tipo 3) então existe um AF $M = (K, T, \delta, S, F)$ tal que $T(M) = L(G)$.
- Seja M um AFND:
  - os estados de M são as variáveis de G, mais um estado adicional A, $A \notin N$. Assim, $K = N \cup \{A\}$;
  - estado inicial de M é S;
  - se P tem produção $S \rightarrow \varepsilon$ então $F = \{S, A\}$, caso contrário $F = \{A\}$;

- transições de M:
  1. $\delta(B, a) = A$ para cada $B \to a \in P$;
  2. $\delta(B, a) = C$ para cada $B \to aC \in P$;
  3. $\delta(A, a) = \emptyset$ para todo $a \in T$.

- Provar que $T(M) = L(G)$.

Exemplo: $G = (\{S, B\}, \{0, 1\}, P, S)$, e

$$P : S \to 0B, B \to 0B|1S|0$$

Construir um AF que aceite L(G):

$$M = (\{S, B, A\}, \{0, 1\}, \delta, S, \{A\})$$

$$\begin{aligned} \delta(S, 0) = \{B\} &\Leftarrow S \to 0B \\ \delta(S, 1) = \emptyset & \\ \delta(B, 0) = \{B, A\} &\Leftarrow B \to 0B|0 \\ \delta(B, 1) = \{S\} &\Leftarrow B \to 1S \end{aligned}$$

Convertendo para um AFD:

$$M' = (K', \{0,1\}, \delta', [S], F')$$

$$K' = \{\emptyset, [S], [A], [B], [AS], [AB], [BS], [ABS]\}$$

$$F' = \{[A], [AS], [AB], [ABS]\}$$

$$\delta'([S], 0) = [B] \quad \delta'([B], 0) = [AB]$$

$$\delta'([S], 1) = \emptyset \quad \delta'([B], 1) = [S]$$

$$\delta'([AB], 0) = \delta(A, 0) \cup \delta(B, 0) = \emptyset \cup \{A, B\} = [AB]$$

$$\delta'([AB], 1) = \delta(A, 1) \cup \delta(B, 1) = \emptyset \cup \{S\} = [S]$$

$$\delta'(\emptyset, 0) = \delta'(\emptyset, 1) = \emptyset$$

Eliminando os estados inacessíveis:

$$K' = \{[S], [B], [AB]\}$$

$$F' = \{[AB]\}$$

- Teorema: Se $M = (K, T, \delta, q_0, F)$ é um AF então existe uma gramática do Tipo 3, $G = (N, T, P, S)$, tal que $L(G) = T(M)$.
- Para obter G:
  - $N = K$;
  - $S = q_0$;
  - Produções:
    1. $B \to aC$ para toda a transição $\delta(B, a) = C$;
    2. $B \to a$ para toda a transição $\delta(B, a) = C$, quando $C \in F$;
  - Se $\varepsilon \notin T(M)$ então $L(G) = T(M)$;

- Se $q_0 \in F$ então $\varepsilon \in T(M)$, e $L(G) = T(M) - \{\varepsilon\}$. Neste caso, construir uma $L(G') = L(G) \cup \{\varepsilon\}$;

• Provar $L(G) = T(M)$.

Exemplo: Contruir $G'' = (K'', \{0,1\}, P'', [S])$,
$K'' = \{[S], [B], [AB]\}$, a partir de M' do exemplo anterior:

$$[S] \to 0[B] \quad \delta'([S], 0) = [B]$$

$$[B] \to 0[AB] \quad \delta'([B], 0) = [AB]$$

$$[B] \to 0 \quad [AB] \in F$$

$$[AB] \to 0[AB] \quad \delta'([AB], 0) = [AB]$$

$$[AB] \to 0 \quad [AB] \in F$$

$$[B] \to 1[S] \quad \delta'([B], 1) = [S]$$

$$[AB] \to 1[S] \quad \delta'([AB], 1) = [S]$$

- Minimização de AF.

  Um AF M é *mínimo* se:

  1. não possui estados inacessíveis;
  2. não possui estados mortos;
  3. não possui estados equivalentes.

- Construção do AF mínimo:
  1. retirar de K os *estados inacessíveis*, ou seja, todo estado $q \in K$ para o qual não exista sentença w tal que $\hat{\delta}(q_0, w) = q$;
  2. retirar de K os *estados mortos*, ou seja, todo estado que não sendo estado final e que não conduz a nenhum estado final durante o reconhecimento;
  3. construir todas as possíveis *classes de equivalência* de estados;

4. construir $M' = (K', \Sigma, \delta', q_0', F')$ conforme segue:
   - K' é o conjunto das classes de equivalência obtidas;
   - q_0' é a classe de equivalência que contém q_0;
   - F' é o conjunto de todas as classes de equivalência que contém pelo menos um estado de F;
   - δ' é o conjunto de transições tais que $\delta'([p], a) = [q]$ para toda a transição $\delta(p_1, a) = q_1$, onde $p_1 \in [p]$ e $q_1 \in [q]$.

- Construção das Classes de Equivalência:

  Um conjunto de estados $q_1, q_2, \ldots, q_j$ está em uma mesma classe de equivalência se todas as transições possíveis a partir de cada estado conduz o autômato aos estados $q_i, q_{i+1}, \ldots, q_n$, estando estes últimos todos em uma mesma classe de equivalência.

- Algoritmo:
  1. Criar um estado $\emptyset$ para representar as transições indefinidas;
  2. Dividir K em duas classes de equivalência iniciais, uma contendo os estados finais e a outra todos os demais estados de K;
  3. Dividir sucessivamente as classes obtidas, até que nenhuma nova classe possa ser obtida.

Exemplo:

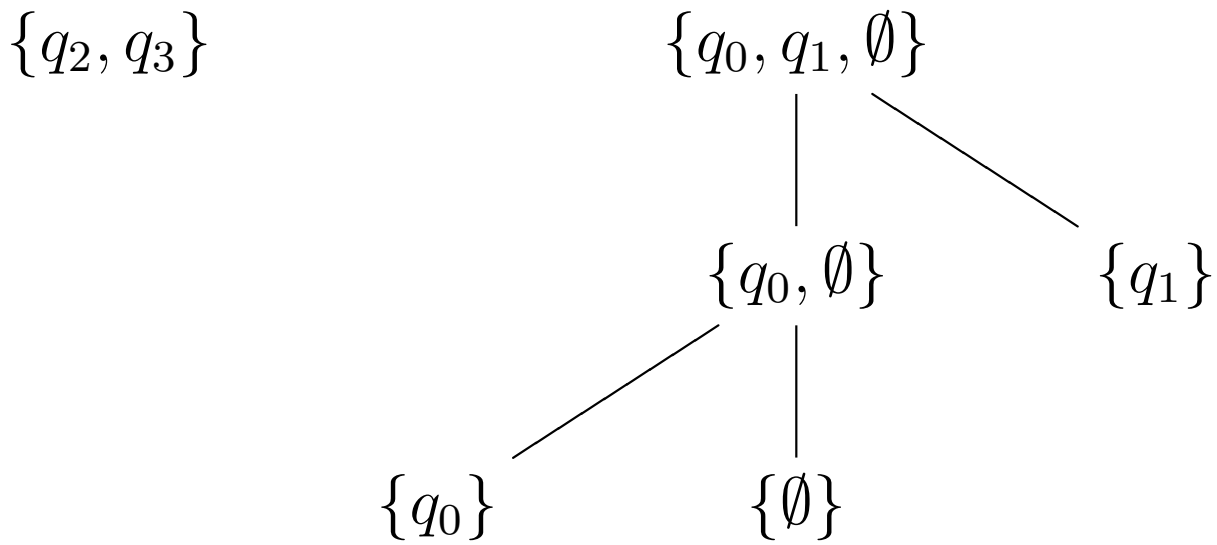
δ	a	b
$\longrightarrow q_0$	q_1	q_5
q_1	-	q_2
$*q_2$	q_3	q_2
$*q_3$	q_3	q_3
q_4	q_4	q_1
q_5	q_5	q_5

Aplicando:

1. estados inacessíveis: q_4;
2. estado morto: q_5;

δ'	a	b
$\rightarrow q_0$	q_1	$\emptyset$
q_1	$\emptyset$	q_2
$*q_2$	q_3	q_2
$*q_3$	q_3	q_3
$\emptyset$	$\emptyset$	$\emptyset$

3. classes de equivalência:



$\{q_2, q_3\}$ $\{q_0\}$ $\{\emptyset\}$ $\{q_1\}$

$[0]$ $[1]$ $[2]$ $[3]$

4. autômato sem classes de equivalência:

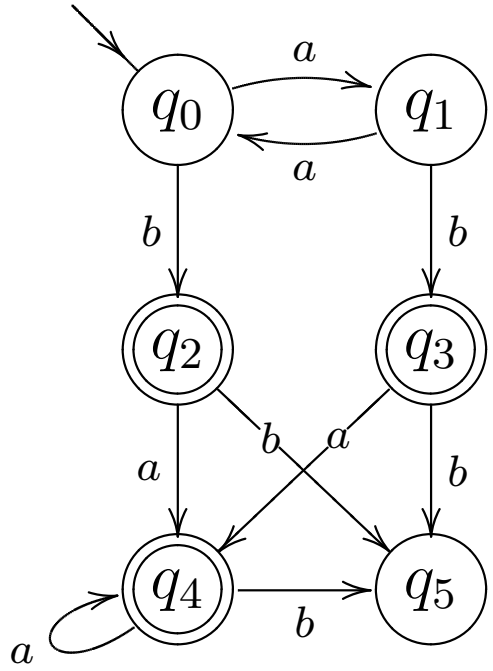
δ''	a	b
$*[0]$	$[0]$	$[0]$
$\rightarrow [1]$	$[3]$	$[2]$
$[2]$	$[2]$	$[2]$
$[3]$	$[2]$	$[0]$

5. AF mínimo:

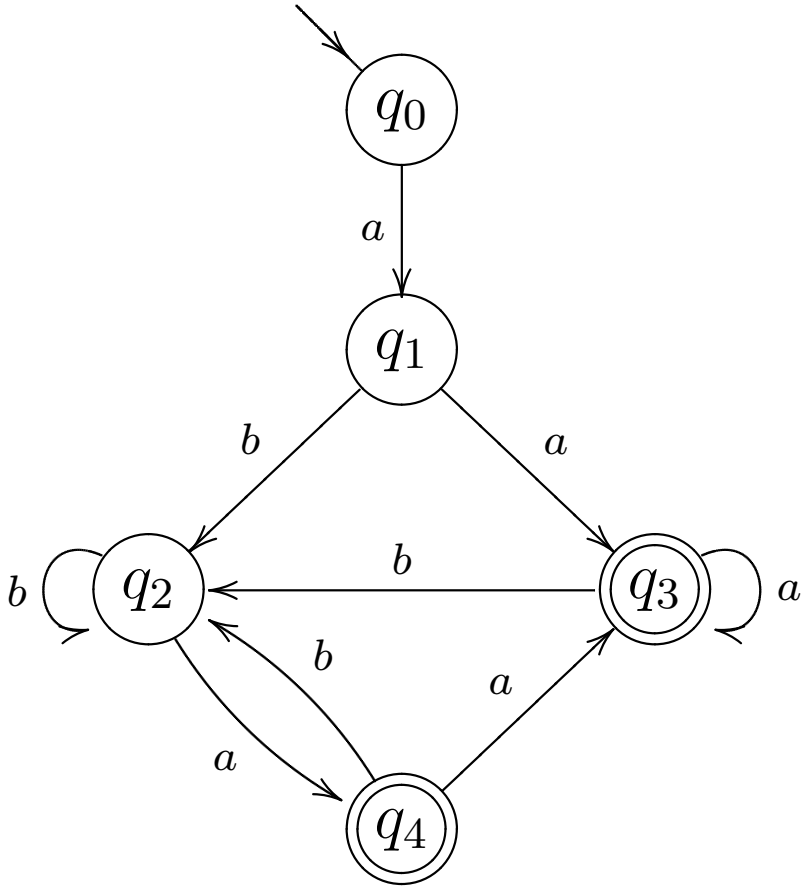
δ'''	a	b
$*[0]$	$[0]$	$[0]$
$\rightarrow [1]$	$[3]$	-
$[3]$	-	$[0]$

Exercícios:

1. Minimize o seguinte AF:



2. Minimize o seguinte AF:



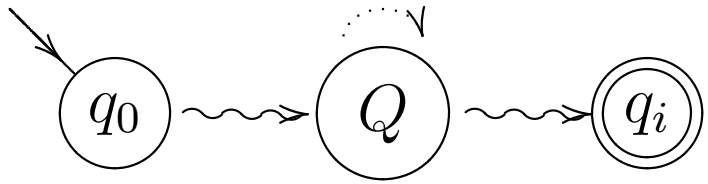
- Expressões Regulares – são esquemas para representar linguagens regulares;
- Sintaxe:
  1. Todos os símbolos do alfabeto são ER;
  2. Se R_1 e R_2 são ER, então $(R_1|R_2)$ é uma ER (união);
  3. Se R_1 e R_2 são ER, então (R_1R_2) é uma ER (concatenação);
  4. Se R_1 é ER, então $(R_1)^*$ é uma ER (repetição).

  Observações: $R_1^+ = R_1(R_1)^*$ e $R_1^? = R_1|\varepsilon$.

- Exemplos:
  - $(a)^* = a^* = \{\varepsilon, a, aa, aaa, \ldots\}$;
  - $a^+ = \{a, aa, aaa, \ldots\}$;
  - $(a|b)^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$;
  - $a|b^* = \{\varepsilon, a, b, bb, bbb, \ldots\}$;
  - $a(a|b)^* = \{a, aa, ab, aaa, aab, aba, abb, aaaa, \ldots\}$;
  - $(a(a|b))^* = \{\varepsilon, aa, ab, aaaa, abaa, aaab, \ldots\}$;
  - Identificadores: $l(l|d)^* = \{l, ll, ld, lll, \ldots\}$;
  - Inteiro com sinal: $(+|-)d^+$;
  - Inteiro com/sem sinal: $((+|-)d^+)|d^+$.

- Bombeamento para Linguagens Regulares (Pumping Lemma) – serve para provar que uma dada linguagem é regular;
- Para provar que *sim*: AF, GR, ER;
- Para provar que *não*: bombeamento;
- Prova por *redução ao absurdo*;
- Seja $M = (K, \Sigma, \delta, q_0, F)$ e $\overline{\overline{K}} = n$;
- Considere uma entrada w tal que $|w| \geq n$;

- Assim, o caminho no diagrama de transições deve passar por um conjunto de estados $Q \subset K$ mais de uma vez (laço);



- Lema: Seja L uma linguagem regular. Então, existe uma constante n tal que se z é uma palavra de L, e $|z| \geq n$, nos podemos dizer que $z = uvw$ de tal forma que $|uv| \leq n$, $|v| \geq 1$ e, para todo $i \geq 0$, uv^iw está em L.

- Aplicação do bombeamento (aplicado como um "jogo" contra um "adversário"):
  1. Selecione uma linguagem L que desejamos provar que não é regular;
  2. O "adversário" escolhe um n, a constante do Lema. Uma vez escolhido o n ele não pode muda-lo;
  3. Selecione uma string $z \in L$. Sua escolha depende do n escolhido no passo anterior;
  4. O "adversário" quebra z nas partes u,v, e w, tal que $|uv| \leq n$ e $|v| \geq 1$;
  5. Voce encontra a contradição mostrando que para qualquer u, v, e w escolhido pelo "adversário", existe um i para o qual uv^iw não está em L.

- Ex: $L = \{a^{n^2} | n \geq 1\}$ não é regular.

  Solução: Suponha $z = a^{n^2}$, $z \in L$ e L é regular. Seja $z = uvw$, onde $1 \leq |v| \leq n$ e $uv^iw \in L$ para todo i. Em particular, seja $i = 2$. Então,
  $n^2 < |uv^2w| \leq n^2 + n$, já que $|uv^2w| = |uvw| + |v|$, $|uvw| = n^2$. Logo, $n^2 < |uv^2w| \leq n^2 + n < (n+1)^2$. Ou seja, o tamanho de uv^2w fica entre n^2 e $(n+1)^2$ e não pode ser um quadrado perfeito. Assim, $uv^2w \notin L$ e L não é regular.

- Exercício: Prove por bombeamento que $L = \{a^n b^n | n \geq 0\}$ não é regular.

Solução: Suponha $z = a^n b^n$, $z \in L$ e L é regular.
Escolho $uv = a^n$, $w = b^n$. Observe que $uv^i w \notin L$ para $i > 1$, já que, neste caso, teriamos $a^m b^n$ para $m > n$.
Logo, por redução ao absurdo, L não é regular.

- AF com Saída (Moore e Mealy) – não se restringe a aceita/rejeita, produz uma string de saída;
- Máquina de Mealy – saída associada às transições;

$$M = (K, \Sigma, \delta, q_0, F, \Delta)$$

Onde:

K conjunto finito de estados;

Σ alfabeto finito de entrada;

$\delta : K \times \Sigma \rightarrow K \times \Delta^*$ função de transição;

$q_0 \in K$ estado inicial;

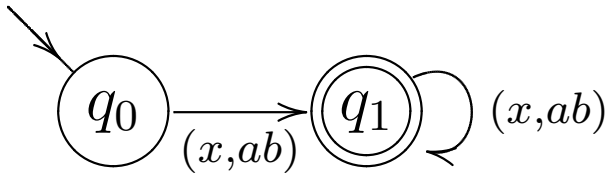
$F \subset K$ conjunto de estados finais;

Δ alfabeto finito de saída;

- $\delta(q, a) = (p, w)$, com $p, q \in K$, $a \in \Sigma$, e $w \in \Delta^*$, significa que o AF, estando no estado q e tendo a na fita de entrada, muda para o estado p e escreve a string w na fita de saída.

Exemplo: Seja $\Sigma = \{x\}$ e $\Delta = \{a, b\}$. M é uma máquina de Mealy que reconhece uma palavra w,$|w| \geq 1$, tal que $w = x^+$ e produz uma seqüência $v = (ab)^+$, em que $|v| = 2|w|$.

Solução:



Observação: arestas são rotuladas com um par (x, y), onde x é o símbolo lido e y é a string produzida na saída.

- Máquina de Moore – saída associada aos estados;

$$M = (K, \Sigma, \delta, q_0, F, \Delta, \delta_S)$$

Onde:

K conjunto finito de estados;

Σ alfabeto finito de entrada;

$\delta : K \times \Sigma \to K$ função de transição;

$q_0 \in K$ estado inicial;

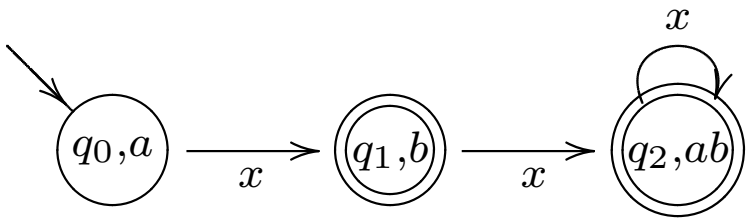
$F \subset K$ conjunto de estados finais;

Δ alfabeto finito de saída;

$\delta_S : K \to \Delta^*$ função (total) de saída;

Exemplo: o mesmo anterior
$(xxx \cdots x \Rightarrow ababab \cdots ab)$.

Solução:



Observação: os vértices são rotulados com um par (x, y), onde x é o nome do estado e y é a saída produzida pelo estado.

- Equivalência das Máquinas de Moore e Mealy
  - Excetuando-se a string vazia, os dois modelos de máquina são equivalentes;
  - A máquina de Moore sempre gera pelo menos a palavra associada ao estado inicial, não podendo assim gerar a sentença vazia;
  - Prova de equivalência por simulação mútua (exceto a sentença vazia);

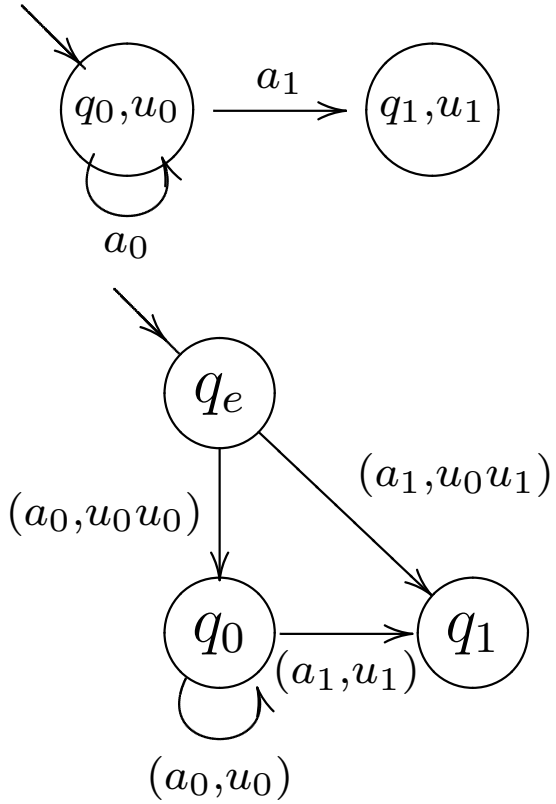
– Teorema: A máquina de Mealy simula a máquina de Moore.

Prova: Suponha MO $= (K, \Sigma, \delta_{\text{MO}}, q_0, F, \delta_S)$ uma máquina de Moore. Seja:

$$\text{ME} = (K \cup \{q_e\}, \Sigma, \delta_{\text{ME}}, q_e, F, \Delta)$$

uma máquina de Mealy, onde δ_{ME} é definido como segue:

1. $\delta_{\text{ME}}(q_e, a) = (\delta_{\text{MO}}(q_0, a), \delta_S(q_0)\delta_S(\delta_{\text{MO}}(q_0, a)))$;
2. $\delta_{\text{ME}}(q, a) = (\delta_{\text{MO}}(q, a), \delta_S(\delta_{\text{MO}}(q, a)))$.


q_0,u_0
a_1
q_1,u_1
a_0
q_e
(a_1,u_0u_1)
(a_0,u_0u_0)
q_0
q_1
(a_1,u_1)
(a_0,u_0)

- Observe que o estado q_e introduzido é referenciado apenas na primeira transição executada. Seu objetivo é garantir a geração da saída referente ao estado inicial q_0 de MO;
- Provar por indução.

– Teorema: A máquina de Moore simula a máquina de Mealy;

Exercícios:

1. Construa um AF M que aceite:
   (a) todas as sentenças em $\{0,1\}^+$ que apresentem cada "1" seguido imediatamente de dois zeros;
   (b) todas as sentenças em $\{a,b\}^*$ de modo que todo "a" apareça entre dois "b";
   (c) todas as sentenças de $\{a,b\}^+$ de modo que o último símbolo seja "b" e o número de símbolos "a" seja par.
2. Construa a gramática regular equivalente a cada um dos autômatos do exercício anterior.

3. Defina a expressão regular que representa valores "redondos" em reais (Ex: R$1,00 ou R$200,00 ou R$1.000.000,00 ou R$3.000,00).

4. Seja a seguinte gramática regular:

$$\begin{aligned} S &\rightarrow 0S|1S|0A|0C|1B \\ A &\rightarrow 0A|0 \\ B &\rightarrow 1B|1 \\ C &\rightarrow 0A|0 \end{aligned}$$

Pede-se:

(a) o AFND M tal que $T(M) = L(G)$;

(b) o AFD M' tal que $T(M) = T(M')$;

(c) M'' tal que $T(M'') = T(M)$ e M'' seja mínimo.

5. Seja G definida por:

$$\begin{aligned} S &\rightarrow aB|aD \\ B &\rightarrow bB|bC \\ C &\rightarrow cC|cD \\ D &\rightarrow d \\ E &\rightarrow aB|a \end{aligned}$$

Pede-se:

(a) o AF M tal que $T(M) = L(G)$;

(b) Determine $T(M)$;

(c) Minimize M.

6. Seja $T(M) = \{ab^n c^m d | 0 \leq n \leq 2 \wedge m \geq 1\}$. Pede-se:
   (a) construa M;
   (b) construa G tal que $L(G) = T(M)$.
7. Construa o AF e a gramática regular equivalentes a cada uma das ER a seguir:
   (a) $a(b|ca)d^*(a|b)^+$;
   (b) $(a(b|ca)^*(b|c)a^*)^+$.

8. Projete uma máquina de Moore que recebendo um valor em dólares (ex: US$10,000.00) converta para reais (suponha relação 1-1 entre dolar e real – logo, R$10.000,00).

9. O mesmo que o exercício anterior, usando máquina de Mealy.

3 Gramática Livre de Contexto e Autômato de Pilha

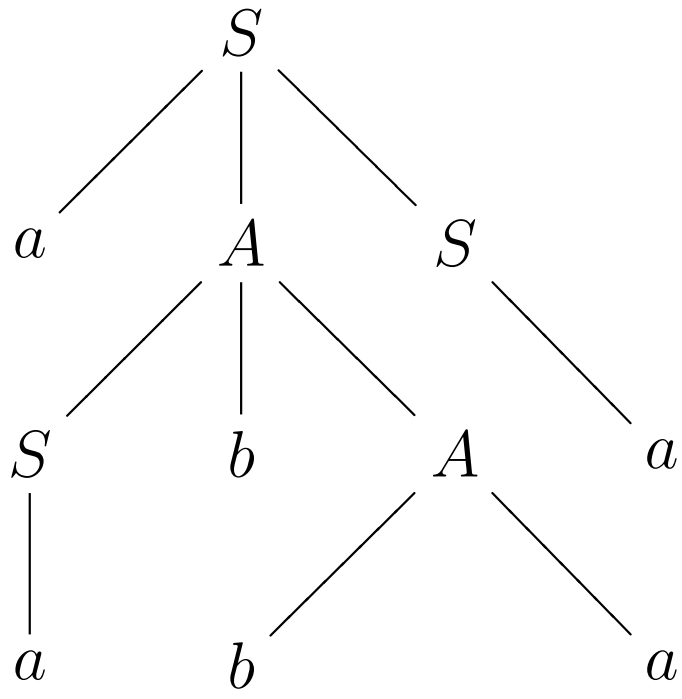
- As GLC tem grande importância pois através delas podemos definir a maioria das estruturas de linguagens de programação;
- As linguagens livres de contexto representam um conjunto mais amplo de linguagens que o das linguagens regulares;
- O autômato de pilha (PDA) é um autômato mais "poderoso" que o AF;
- AF $\sqsubseteq$ PDA (PDA simula AF e AF não simula PDA);

- Extenção da definição de GLC para permitir produções na forma $A \rightarrow \varepsilon$ (chamadas ε-*produções*);
- Nova definição: $P = \{A \rightarrow \beta | A \in N \wedge \beta \in (N \cup T)^*\}$;
- Uma GLC é ε-*livre* quando não possui ε-produções ou quando possui apenas $S \rightarrow \varepsilon$, onde S é o símbolo inicial da gramática, e S não aparece do lado direito de nenhuma regra de produção;

- Árvores de Derivação para GLC's – são representações gráficas para as derivações da GLC (formam uma estrutura hierárquica);

- Seja $G = (N, T, P, S)$ uma GLC. Uma árvore é uma *árvore de derivação* para G se:
  1. todo nodo tem um rótulo que é um símbolo de V;
  2. o rótulo da raiz é S;
  3. se um nodo A tem um ou mais descendentes, então A é um elemento de N;
  4. se $A_1, A_2, \ldots, A_n$ são descendentes diretos de A, da esquerda para a direita, então $A \to A_1 A_2 \cdots A_n \in P$;
  5. se D é a única sub-árvore da raiz e tem rótulo ε, então $S \to \varepsilon \in P$.

- Exemplo: $G = (\{S, A\}, \{a, b\}, P, S)$,
  $P = \{S \to a|aAS, A \to SbA|SS|ba\}$. Derivação:
  $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$.
  Árvore de derivação:



- *Profundidade da árvore de derivação* é o comprimento do maior caminho entre a raiz e um nodo terminal. No exemplo anterior é 3;
- *Limite de uma árvore de derivação* é a seqüência formada pela concatenação, da esquerda para a direita, das folhas da árvore de derivação;

- Derivação mais à esquerda e mais à direita – a árvore de derivação ignora variações na ordem em que os símbolos são substituidos na derivação. Assim,

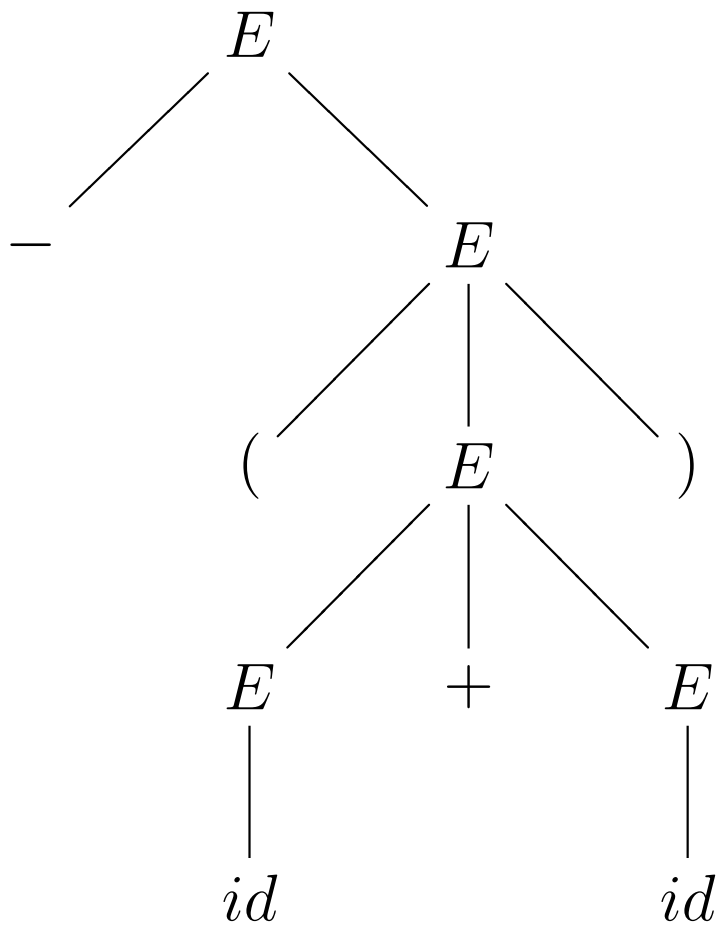
$$G = (\{E\}, \{+, *, (,), -, id\}, P, S)$$

$$P = \{E \rightarrow E + E | E * E | (E) | - E | id\}$$

tomando a sentença $-(id + id)$ podemos deriva-la das duas seguintes formas:

1. $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(id + E) \Rightarrow -(id + id)$;
2. $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(E + id) \Rightarrow -(id + id)$;

Ambas correspondem a mesma árvore de derivação:



- Derivação mais à esquerda: símbolo substituido é sempre o não-terminal mais à esquerda na forma sentencial;
- Derivação mais à direita: símbolo substituido é sempre o não-terminal mais à direita na forma sentencial;
- Nas duas derivações mostradas a pouco, a primeira é mais à esquerda e a segunda mais à direita;

- Exemplo: $id + id * id$

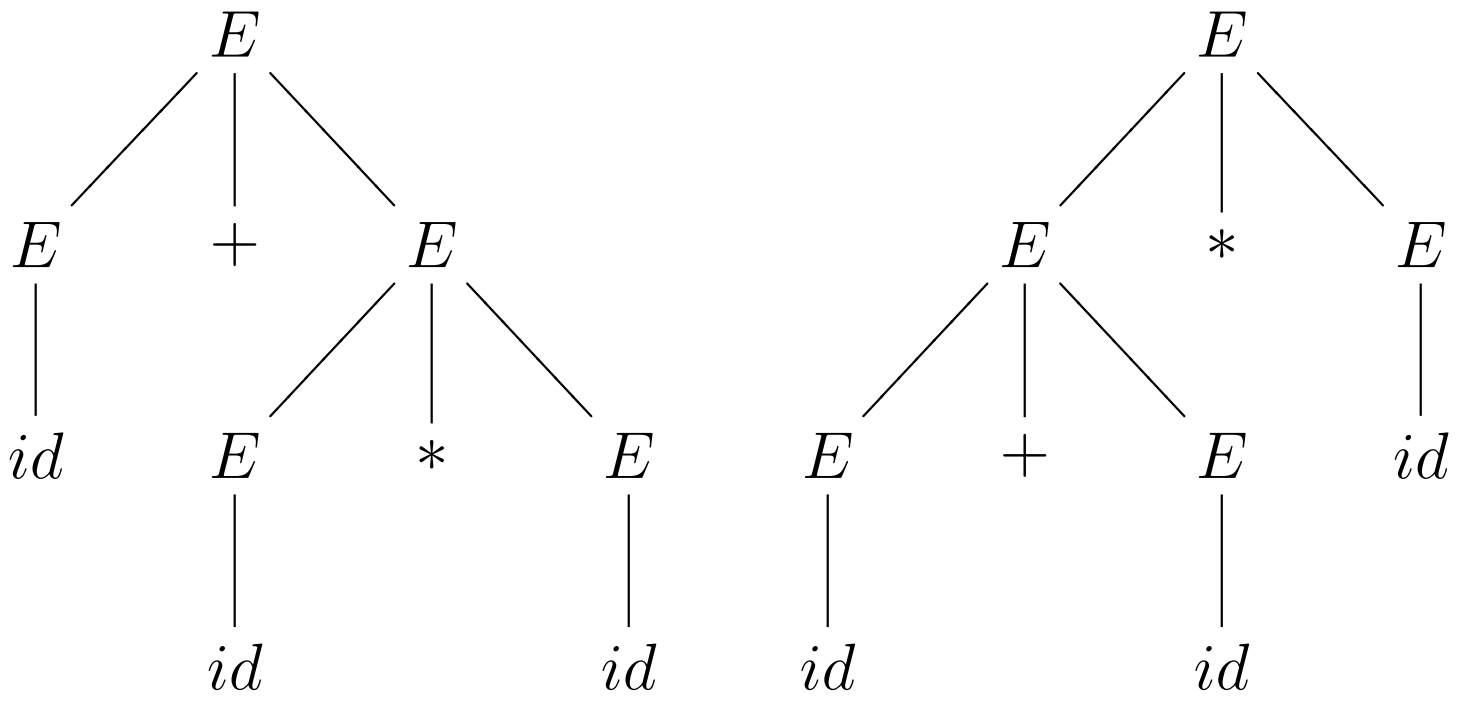
  Derivação mais à esquerda: $E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$

  Derivação mais à direita: $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * id \Rightarrow E + id * id \Rightarrow id + id * id$

  Observação: se $w \in L(G)$ e G é uma GLC, então w tem pelo menos uma árvore de derivação e, correspondendo a esta árvore existe uma só derivação mais à esquerda e uma só derivação mais à direita.

- Uma GLC é *ambígua* se para a mesma sentença existe mais de uma árvore de derivação;

  Exemplo: $id + id * id$



- Uma *linguagem inerentemente ambígua* é aquele em que todas as GLC que a geram são ambíguas.

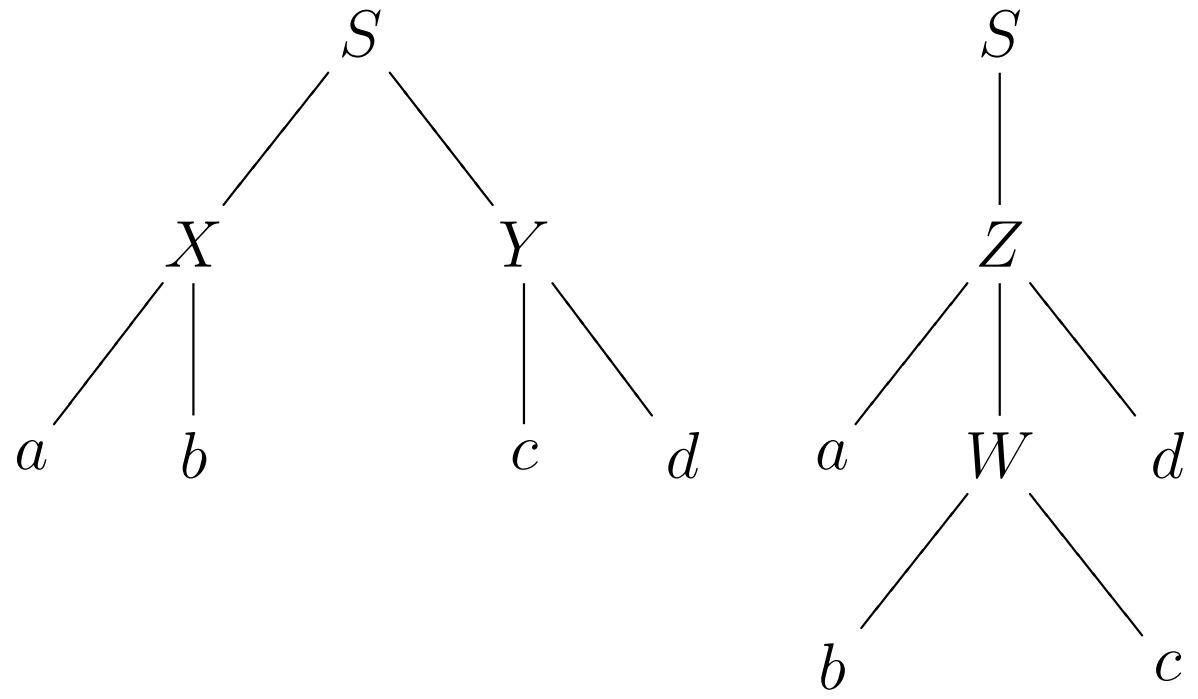
  Exemplo: $L = \{a^n b^n c^m d^m | n \geq 1 \wedge m \geq 1\} \bigcup \{a^n b^m c^m d^n | n \geq 1 \wedge m \geq 1\}$

Representada pela seguinte gramática:

$$G = (\{S, X, Y, Z, W\}, \{a, b, c, d\}, P, S)$$

$$\begin{aligned} S &\rightarrow XY|Z \\ X &\rightarrow ab|aXb \\ Y &\rightarrow cd|cXd \\ Z &\rightarrow aWd|aZd \\ W &\rightarrow bc|bWc \end{aligned}$$

Exemplo de derivação: $abcd$



- Transformações de GLC – tem o objetivo de simplificação e preparação para aplicações posteriores. Observação: a gramática transformada deve ser equivalente à original.

- Eliminação de símbolos inúteis

  Um símbolo é *inútil* se ele não aparece na derivação de nenhuma sentença.

  estéril não gera nenhuma seqüência de terminais pertencente a uma sentença;

  inalcançável não aparece em nenhuma forma sentencial da gramática.

- Determinação do conjunto de símbolos férteis:

  1. Construir o conjunto $N_0 = \emptyset$ e fazer $i = 1$;
  2. Repetir
     (a) $N_i = \{A | A \rightarrow \alpha \in P \wedge \alpha \in (N_{i-1} \cup T)^*\}$;
     (b) $i = i + 1$;

     até que $N_i = N_{i-1}$;
  3. N_i é o conjunto de símbolos férteis.

- Observação: se o símbolo inicial não fizer parte dos símbolos férteis, então a linguagem gerada pela gramática é *vazia*.

- Exemplo: $G = (\{S, A, B, C, D\}, \{a, b, c, d\}, P, S)$, $P = \{S \to aA, A \to a|bB, B \to b|dD, C \to cC|c, D \to dD\}$. Solução:

  1. $N_0 = \emptyset$;
  2. $N_1 = \{A, B, C\}$;
  3. $N_2 = \{S, A, B, C\}$;
  4. $N_3 = \{S, A, B, C\} = N_2$.

  Conjunto dos símbolos férteis: $\{S, A, B, C\}$.

  Gramática simplificada:
  $G' = (\{S, A, B, C\}, \{a, b, c\}, P', S)$,
  $P' = \{S \to aA, A \to a|bB, B \to b, C \to cC|c\}$.

- Determinação do conjunto de símbolos alcançáveis:

  1. Construir o conjunto $V_0 = \{S\}$ (S=símbolo inicial) e fazer $i = 1$;

  2. Repetir

     (a) $V_i = \{X | \exists A \rightarrow \alpha X \beta \in P \wedge A \in V_{i-1} \wedge \alpha, \beta \in (N \cup T)^* \wedge X \in V\} \bigcup V_{i-1}$;

     (b) $i = i + 1$;

     até que $V_i = V_{i-1}$;

  3. V_i é o conjunto de símbolos alcançáveis.

- Exemplo: G' do exemplo anterior.
  $G' = (\{S, A, B, C\}, \{a, b, c\}, P', S)$,
  $P' = \{S \to aA, A \to a|bB, B \to b, C \to cC|c\}$.
  Solução:
  1. $V_0 = \{S\}$;
  2. $V_1 = \{a, A\} \bigcup V_0 = \{S, a, A\}$;
  3. $V_2 = \{a, b, B\} \bigcup V_1 = \{S, a, A, b, B\}$;
  4. $V_3 = \{b\} \bigcup V_2 = \{S, a, A, b, B\} = V_2$.

  Conjunto de símbolos alcançáveis: $\{S, A, B, a, b\}$.
  Gramática simplificada:
  $G'' = (\{S, A, B\}, \{a, b\}, P'', S)$,
  $P'' = \{S \to aA, A \to a|bB, B \to b\}$.

- Transformação de GLC qualquer em GLC ε-livre:
  1. Reunir em um conjunto os não-terminais que derivam direta ou indiretamente a sentença vazia: $N_e = \{A | A \in N \wedge A \Rightarrow^+ \varepsilon\}$;
  2. Construir o conjunto de regras P' como segue:
     (a) incluir em P' todas as regras de P, com excessão daquelas da forma $A \rightarrow \varepsilon$;
     (b) para cada ocorrência de um símbolo N_e do lado direito de alguma regra de P, incluir em P' mais uma regra, substituindo este símbolo por ε. Isto é, para cada regra de P tipo $A \rightarrow \alpha B \beta$, $B \in N_e$ e $\alpha, \beta \in V^*$, incluir em P' a regra $A \rightarrow \alpha\beta$;

3. Se $S \in N_e$, adicionar a P as regras $S' \rightarrow S$ e $S' \rightarrow \varepsilon$, incluindo este novo não-terminal S' em $N' = N \cup \{S'\}$. Caso contrário, trocar os nomes de S por S' e de N por N';
4. A nova gramática será definida por: $G = (N', T, P', S')$.

- Exemplos:
  1. $P : S \to aB, B \to bB|\varepsilon$. Solução: $N_e = \{B\}$ e
     $P' : S \to aB|a, B \to bB|b$;
  2. $P : S \to bDCe, D \to dD|\varepsilon, C \to cC|\varepsilon$. Solução:
     $N_e = \{D, C\}$ e
     $P' : S \to bDCe|bCe|bDe|be, D \to dD|d, C \to cC|c$;
  3. $P : S \to aS|\varepsilon$. Solução: $N_e = \{S\}$ e
     $P' : S' \to S|\varepsilon, S \to aS|a$.

- Remoção de Produções Simples – *produções simples* são produções na forma $A \rightarrow B$, onde $A, B \in N$:
  1. Transformar a GLC em GLC ε-livre, se necessário;
  2. Para todo não-terminal de N, construir um conjunto com os não-terminais que ele pode derivar em zero ou mais passos. Isto é, $\forall A \in N$, construir $N_A = \{B | A \Rightarrow^* B\}$;
  3. Se $B \rightarrow \alpha \in P$ e não é produção simples, adicione a P' as produções $A \rightarrow \alpha$ para todo A tal que $B \in N_A$;
  4. a GLC equivalente sem produções simples é $G = (N, T, P', S)$.

- Exemplos:
  1. $P : S \to bS|A, A \to aA|a$. Solução: $N_S = \{S, A\}$, $N_A = \{A\}$, e $P' : S \to bS|aA|a, A \to aA|a$;
  2. $P : S \to aSb|A, A \to aA|B, B \to bBc|bc$. Solução: $N_S = \{S, A, B\}$, $N_A = \{A, B\}$, $N_B = \{B\}$, e $P' : S \to aSb|aA|bBc|bc, A \to aA|bBc|bc$, $B \to bBc|bc$.

- Fatoração de GLC

  Uma GLC está fatorada se ela é determinística, isto é, não possui produções cujo lado direito inicie com o mesmo conjunto de símbolos ou com símbolos que gerem seqüências que iniciem com o mesmo conjunto de símbolos. Por exemplo, uma gramática fatorada não poderia apresentar as seguintes regras:

  $$A \to aB|aC\,,$$

  pois ambas inicial com o terminal a.

  Outro exemplo de gramática não-fatorada é:

  $$S \to A|B \quad A \to ac \quad B \to ab\,.$$

- Para fatorar:
  1. as produções que apresentam não-determinismo direto, da forma $A \rightarrow \alpha\beta|\alpha\gamma$ serão substituidas por

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta|\gamma \end{aligned}$$

     sendo A' um novo não-terminal;
  2. o não-determinismo indireto é retirado fazendo, nas regras de produção, as derivações necessárias para torna-lo um não-determinismo direto, resolvido com o passo anterior.

Exemplos:

1. $P : S \to aA|aB, A \to aA|a, B \to b$. Solução:
   $P' : S \to aS', S' \to A|B, A \to aA', A' \to A|\varepsilon, B \to b$;
2. $P : S \to Ab|ab|baA, A \to aab|b$. Solução:
   (não-determinismo indireto)
   $P' : S \to aabb|bb|ab|baA, A \to aab|b$ e
   $P'' : S \to aS'|bS'', S' \to abb|b, S'' \to b|aA, A \to aab|b$

- Fatoração é importante pois na implementação de um compilador, o mesmo deve seguir uma gramática que não apresente não-determinismo pois, caso contrário, no processo de reconhecimento haveria "retornos" (backtracking) que acabam reduzindo a eficiência do algoritmo de reconhecimento.

- Teorema: Toda LLC sem ε pode ser definida por uma gramática que não contém símbolos inúteis, ε-produções, nem produções simples.
- Prova: Conseqüência dos algoritmos apresentados.

- Eliminação da Recursão à Esquerda

  Um não-terminal A é *recursivo* se $A \Rightarrow^+ \alpha A \beta$, $\alpha, \beta \in V^*$. Se $\alpha = \varepsilon$, então A é recursivo à esquerda. Se $\beta = \varepsilon$, é recursivo à direita. A recursividade pode ser direta ou indireta.

  Uma gramática é *recursiva à esquerda* se possui pelo menos um não-terminal recursivo à esquerda. Se possui pelo menos um não-terminal recursivo à direita, ela é chamada *recursiva à direita*.

- A importância da recursividade à esquerda é que alguns tipos de compiladores podem executar o processo de reconhecimento como chamadas de rotinas (procedimentos ou funções). Assim, uma gramática recursiva à esquerda, tal como por exemplo $A \rightarrow Aa|a$, acaba gerando um laço infinito $A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow \cdots$ e o processo de reconhecimento não finaliza nunca.

- Algoritmo:

  1. recursões diretas: substituir cada regra $A \rightarrow A\alpha_1|A\alpha_2|\cdots|A\alpha_n|\beta_1|\beta_2|\cdots|\beta_m$, onde nenhum β_i começa por A, por:

$$\begin{aligned} A &\rightarrow \beta_1 A'|\beta_2 A'|\cdots|\beta_m A' \\ A' &\rightarrow \alpha_1 A'|\alpha_2 A'|\cdots|\alpha_n A'|\varepsilon \end{aligned}$$

2. recursões indiretas:

   (a) ordene os não-terminais de G em uma ordem qualquer $(A_1, A_2, \ldots, A_n)$;

   (b)

   para i de 1 até n faça
   para j de 1 até $(i-1)$ faça
   troque $A_i \rightarrow A_j\gamma$ por $A_i \rightarrow \delta_1\gamma|\delta_2\gamma|\cdots|\delta_k\gamma$, onde $\delta_1, \delta_2, \ldots, \delta_k$ são os lados direitos das A_j-produções (ou seja, $A_j \rightarrow \delta_1|\cdots|\delta_k$);
   fim_para
   elimine as recursões diretas das A_i-produções;
   fim_para.

- Exemplo: $P : S \to Aa, A \to Sb|cA|a$. Solução:
  $(A_1 = S, A_2 = A)$, $n = 2$, e

  1. $i = 1$ (não faz j pois o laço é de 1 até 0);

     Eliminando as recursões diretas das S-produções:
     não faz nada (pois não tem);

  2. $i = 2$ e $j = 1$:

     Trocar produções tipo $A \to S\gamma$:
     $P' : S \to Aa, A \to Aab|cA|a$;

     Eliminar as recursões diretas das A-produções:
     $P'' : S \to Aa, A \to cAA'|aA', A' \to abA'|\varepsilon$.

- Exercício: Elimine a recursão à esquerda da GLC $P : S \to Aa|b, A \to Bb|a, B \to Sb|b$.

- Forma Normal de Chomsky (FNC)

  A FNC é uma *forma canônica*. Uma GLC está na FNC se ela é ε-livre e apresenta todas as produções da forma

  $$A \to BC \quad \text{ou} \quad A \to a\,,$$

  onde $A, B, C \in N$ e $a \in T$.

- Teorema: Toda LLC ε-livre pode ser gerada por uma GLC na FNC.

Para converter uma GLC $G = (N, T, P, S)$ ε-livre para a FNC:

1. obter $G' = (N', T, P', S)$ a partir de G, removendo de G as produções simples, de modo que $L(G') = L(G)$;
2. nas regras de G' em que o lado direito apresenta mais de um termo, substituir cada terminal $a \in T$ por um novo não-terminal A_a, incluindo para cada destes novos não-terminais uma nova regra $A_a \rightarrow a$, resultando em $G'' = (N'', T, P'', S)$;

3. substituir cada regra do tipo

$$A \to B_1 B_2 \cdots B_m \,, \quad m \geq 3$$

onde $A, B_1, \ldots B_m$ são não-terminais, pelo conjunto de regras:

$$\begin{aligned} A &\to B_1 B_1' \\ B_1' &\to B_2 B_2' \\ &\vdots \\ B_{m-2}' &\to B_{m-1} B_m \end{aligned}$$

onde $B_1', B_2', \ldots, B_{m-2}'$ são novos não-terminais;

4. a gramática na FNC é $G''' = (N''', T, P''', S)$.

- Exemplo: Obtenha a FNC.

$$\begin{aligned} P: S &\rightarrow A|ABA \\ A &\rightarrow aA|a \\ B &\rightarrow bB|b \end{aligned}$$

Solução:

1.

$$\begin{aligned} P': S &\rightarrow aA|a|ABA \\ A &\rightarrow aA|a \\ B &\rightarrow bB|b \end{aligned}$$

2.

$$
\begin{aligned}
P'' : S &\rightarrow A_a A | a | ABA \\
A &\rightarrow A_a A | a \\
B &\rightarrow A_b B | b \\
A_a &\rightarrow a \\
A_b &\rightarrow b
\end{aligned}
$$

3.

$$
\begin{aligned}
P''' : S &\rightarrow A_a A | a | A B' \\
B' &\rightarrow B A \\
A &\rightarrow A_a A | a \\
B &\rightarrow A_b B | b \\
A_a &\rightarrow a \\
A_b &\rightarrow b
\end{aligned}
$$

- Forma Normal de Greibach (FNG)

  A FNG é também uma forma canônica. Uma GLC está na FNG se ela é ε-livre e apresenta todas as produções na forma:

  $$A \to a\alpha\,,$$

  onde $A \in N$, $a \in T$, e $\alpha \in N^*$.

- Teorema: Toda LLC ε-livre pode ser gerada por uma GLC na FNG.

- Algoritmo:
  1. achar $G' = (N', T, P', S)$ tal que $L(G') = L(G)$ e G' está na FNC;
  2. ordenar os não-terminais de G' em uma ordem qualquer $N' = (A_1, A_2, \ldots, A_m)$;
  3. modificar as regras de P' de modo que, se $A_i \rightarrow A_j\gamma \in P'$, então $j > i$;
  4. a gramática obtida no passo anterior, G'', apresentará todas as regras de A_m com o lado direito iniciando por terminal. Por substituições sucessivas dos primeiros termos das regras A_i anteriores, coloca-se estas também na mesma forma;

5. se no item 3 tiverem sido incluídos novos não-terminais B_i (para retirar recursões à esquerda), fazer também para as regras correspondentes a estes, as devidas substituições dos primeiros termos (que serão sempre terminais ou A_i);
6. A G''' está na FNG.

- Exemplo: Obtenha a FNG.

$$\begin{aligned} P : S &\rightarrow AS|a \\ A &\rightarrow SA|b \end{aligned}$$

Solução:

1. G já está na FNC;
2. Renomear os não-terminais: $S = A_1$ e $A = A_2$;

$$\begin{aligned} P : A_1 &\rightarrow A_2A_1|a \\ A_2 &\rightarrow A_1A_2|b \end{aligned}$$

3. se $A_i \to A_j\gamma \in P'$, então $j > i$. A única regra a modificar é:

$$A_2 \to A_1 A_2 \,.$$

Substituindo A_1 nesta regra:

$$A_2 \to A_2 A_1 A_2 | a A_2 | b \,.$$

Retirando a recursão à esquerda de

$$\overbrace{A_2}^{A} \rightarrow \overbrace{A_2}^{A}\overbrace{A_1A_2}^{\alpha_1} | \overbrace{aA_2}^{\beta_1} | \overbrace{b}^{\beta_2},$$

obteremos:

$$\begin{aligned}
P'' : A_1 &\rightarrow A_2A_1|a \\
\overbrace{A_2}^{A} &\rightarrow \overbrace{aA_2}^{\beta_1}\overbrace{B_2}^{A'} | \overbrace{b}^{\beta_2}\overbrace{B_2}^{A'} \\
\overbrace{B_2}^{A'} &\rightarrow \overbrace{A_1A_2}^{\alpha_1}\overbrace{B_2}^{A'} |\varepsilon
\end{aligned}$$

Retirando o ε por substituições:

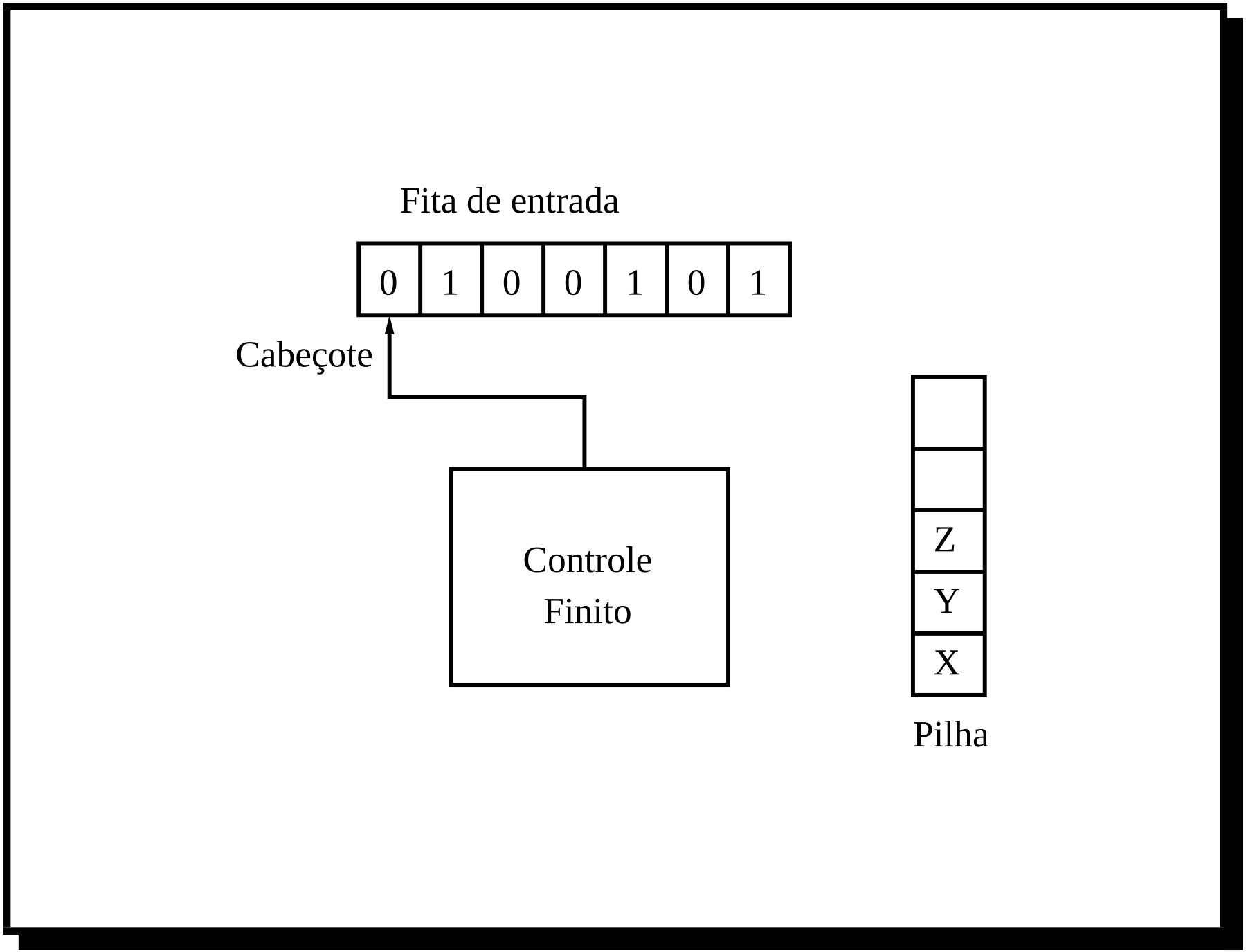
$$\begin{aligned} P'' : A_1 &\rightarrow A_2A_1|a \\ A_2 &\rightarrow aA_2B_2|bB_2|aA_2|b \\ B_2 &\rightarrow A_1A_2B_2|A_1A_2 \end{aligned}$$

4. Fazendo as substituições finais para tornar todos os lados direitos na forma $A \to a\alpha$, $A \in N$, $a \in T$ e $\alpha \in N^*$:

$$
\begin{aligned}
P''' : A_1 &\to aA_2B_2A_1|bB_2A_1|aA_2A_1|bA_1|a \\
A_2 &\to aA_2B_2|bB_2|aA_2|b \\
B_2 &\to aA_2B_2A_1A_2B_2|bB_2A_1A_2B_2| \\
&\quad aA_2A_1A_2B_2|bA_1A_2B_2|aA_2B_2| \\
&\quad aA_2B_2A_1A_2|bB_2A_1A_2|aA_2A_1A_2| \\
&\quad bA_1A_2|aA_2
\end{aligned}
$$

- Gramática de Operadores – é uma GLC que não possui produções $A \rightarrow \alpha BC\beta$, onde $A, B, C \in N$, e $\alpha, \beta \in V^*$. Ou seja, uma GLC em que não aparecem dois não-terminais juntos em um lado direito. Ex: $P : E \rightarrow E + E|E - E|E * E|id$. Importância: expressões aritméticas de linguagens de programação.

- Autômato de Pilha
  - Máquina abstrata que reconhece as LLC;
  - Chamado também de "Push-down Automata" (PDA);
  - Consiste de:
    - controle finito;
    - fita de entrada;
    - pilha.
  - Característica: ser não-determinístico.


Fita de entrada
0
1
0
0
1
0
1
Cabeçote
Controle
Finito
Z
Y
X
Pilha

- Movimentos do PDA:
  1. Um símbolo é lido e o cabeçote avança para o próximo símbolo;
  2. "ε-move" – movimento vazio, no qual o cabeçote não se move, não importando qual é o símbolo que está sendo apontado na fita de entrada.

- Linguagem Aceita por um PDA

  Dois tipos de reconhecimento:

  1. pilha vazia;
  2. estado final.

  Observações:

  - No reconhecimento por pilha vazia o conjunto de estados finais é irrelevante;
  - Os dois tipos são equivalentes (resultam na mesma classe de linguagens);
  - Classe reconhecida: LLC.

- Definição Formal de PDA

$$M = (K, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Onde:

K conjunto finito de estados;

Σ alfabeto finito de entrada;

Γ alfabeto finito da pilha;

$\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \text{Partes}(K \times \Gamma^*)$ função de transição (mapeamentos);

$q_0 \in K$ estado inicial;

$z_0 \in \Gamma$ símbolo inicial da pilha;

$F \subseteq K$ conjunto finito de estados finais.

- Tipos de mapeamentos:
  1. $\delta(q, a, z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$, onde $q, p_1, \ldots, p_m \in K$, $a \in \Sigma$, $z \in \Gamma$, e $\gamma_1, \ldots, \gamma_m \in \Gamma^*$.
     Significado:
     - M está no estado p, com a na entrada e z no topo da pilha;
     - M pode mudar para o estado p_i, $1 \leq i \leq m$, substituindo z por γ_i no topo da pilha;
     - M avança o cabeçote para o próximo símbolo.

2. $\delta(q, \varepsilon, z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$.
   Significado:
   - M está no estado p, com z no topo da pilha;
   - não interessa o que está na fita de entrada;
   - M passa ao estado p_i, $1 \leq i \leq m$, trocando z por γ_i no topo da pilha;
   - o cabeçote não se move (movimento vazio);

- Dependendo de $|\gamma|$ temos as seguintes possíveis ações do PDA em relação à pilha:
  1. $|\gamma| \geq 2$ – troca o símbolo no topo da pilha e empilha outros (a pilha cresce);
  2. $|\gamma| = 1$ – troca o símbolo no topo (pilha permanece com o mesmo tamanho);
  3. $|\gamma| = 0$ ($\gamma = \varepsilon$) – desempilha o símbolo do topo (a pilha decresce de tamanho).

- Convenção: costuma-se representar a pilha na horizontal, com o topo da pilha sendo aquele símbolo que está mais à esquerda.

- Um autômato de pilha é *determinístico* se:
  1. para qualquer $q \in K$, $z \in \Gamma$ e $a \in \Sigma \cup \{\varepsilon\}$, $\delta(q, a, z)$ nunca contém mais de um elemento;
  2. para cada $q \in K$ e $z \in \Gamma$, sempre que $\delta(q, \varepsilon, z) \neq \emptyset$ então $\delta(q, a, z) = \emptyset$ para todo $a \in \Sigma$.

  Observação: o segundo item impede que ocorra a escolha entre um movimento envolvendo um símbolo na entrada e um "ε-move".

- Exemplo: PDA que aceita, por pilha vazia,

$$L = \{wcw^R | w \in \{0,1\}^*\},$$

onde w^R é a seqüencia reversa de w.

$$M = (\{q_1, q_2\}, \{0, 1, c\}, \{z_0, z, u\}, \delta, q_1, z_0, \emptyset)$$

$$\delta(q_1, 0, z_0) = \{(q_1, zz_0)\} \quad \delta(q_1, 0, z) = \{(q_1, zz)\}$$

$$\delta(q_1, 0, u) = \{(q_1, zu)\} \quad \delta(q_1, c, z_0) = \{(q_2, z_0)\}$$

$$\delta(q_1, c, z) = \{(q_2, z)\} \quad \delta(q_1, c, u) = \{(q_2, u)\}$$

$$\delta(q_2, 0, z) = \{(q_2, \varepsilon)\} \quad \delta(q_2, \varepsilon, z_0) = \{(q_2, \varepsilon)\}$$

$$\delta(q_1, 1, z_0) = \{(q_1, uz_0)\} \quad \delta(q_1, 1, z) = \{(q_1, uz)\}$$

$$\delta(q_1, 1, u) = \{(q_1, uu)\} \quad \delta(q_2, 1, u) = \{(q_2, \varepsilon)\}$$

- Chama-se *configuração* de um PDA a um par (q, γ) onde $q \in K$ e $\gamma \in \Gamma^*$, significando que o PDA está no estado q, com γ armazenado na pilha;
- passagem de uma configuração para outra:

$$a : (q, z\gamma) \vdash_M (p, \beta\gamma),$$

onde: $a \in \Sigma \cup \{\varepsilon\}$, $\gamma, \beta \in \Gamma^*$, $z \in \Gamma$, e $(p, \beta) \in \delta(q, a, z)$;

- Extendendo para seqüências:

$$a_1 a_2 \cdots a_n : (q_1, \gamma_1) \vdash_M^* (q_{n+1}, \gamma_{n+1}),$$

onde: $a_1, a_2, \ldots, a_n \in \Sigma \bigcup \{\varepsilon\}$, $q_1, q_2, \ldots, q_{n+1} \in K$, $\gamma_1, \gamma_2, \ldots, \gamma_{n+1} \in \Gamma^*$, e $a_i : (q_i, \gamma_i) \vdash_M (q_{i+1}, \gamma_{i+1})$, para todo $1 \leq i \leq n$;

- convenção: $\varepsilon : (q, \gamma) \vdash_M^* (q, \gamma)$.

- A *descrição instantânea* de um PDA é formada pelo seu estado atual, o conteúdo da pilha e o restante da seqüência de entrada a ser lida. Representa a situação do reconhecimento em um determinado instante;
- Convenção:

$$(q, bc, xyzz_0)\,,$$

significando que o autômato está no estado q, com bc restando na entrada (cabeçote esta sobre o símbolo b), e a pilha contém $xyzz_0$, com x no topo.

- Exemplo: reconhecimento da sentença $01c10$:

$$\begin{aligned}
(q_1, 01c10, z_0) &\vdash (q_1, 1c10, zz_0) \vdash \\
&\vdash (q_1, c10, uzz_0) \vdash \\
&\vdash (q_2, 10, uzz_0) \vdash \\
&\vdash (q_2, 0, zz_0) \vdash \\
&\vdash (q_2, \varepsilon, z_0) \vdash \\
&\vdash (q_2, \varepsilon, \varepsilon) \quad \text{(pilha vazia)}
\end{aligned}$$

Produções usadas: $\delta(q_1, 0, z_0) = (q_1, zz_0)$,
$\delta(q_1, 1, z) = (q_1, uz)$, $\delta(q_1, c, u) = (q_2, u)$,
$\delta(q_2, 1, u) = (q_2, \varepsilon)$, $\delta(q_2, 0, z) = (q_2, \varepsilon)$,
$\delta(q_2, \varepsilon, z_0) = (q_2, \varepsilon)$.

- Linguagem aceita por um PDA
  - $T(M)$ – linguagem aceita por estado final:

$$T(M) = \{w | w : (q_0, z_0) \vdash_M^* (q, \gamma), \gamma \in \Gamma^*, q \in F\};$$

  - $N(M)$ – linguagem aceita por pilha vazia:

$$N(M) = \{w | w : (q_0, z_0) \vdash_M^* (q, \varepsilon), q \in K\}.$$

- Exemplo: M, não-determinístico, aceitando por pilha vazia

$$N(M) = \{ww^R | w \in \{0,1\}^*\}\,.$$

Solução: $M = (\{q_1, q_2\}, \{0, 1\}, \{z_0, z, u\}, \delta, q_1, z_0, \emptyset)$

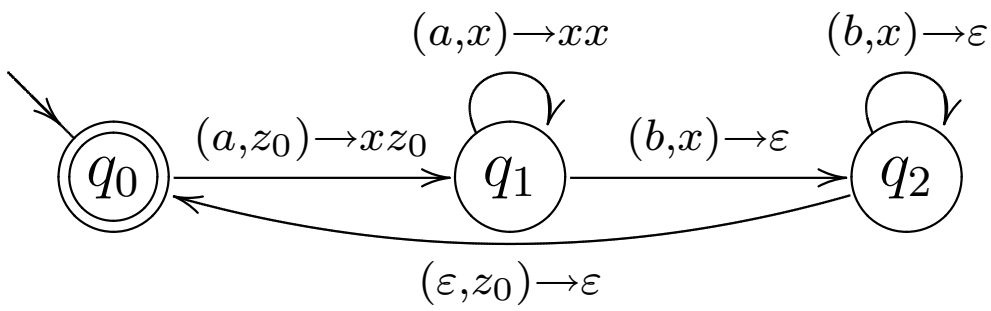
$$
\begin{aligned}
\delta(q_1, 0, z_0) &= \{(q_1, zz_0)\} \\
\delta(q_1, 1, z_0) &= \{(q_1, uz_0)\} \\
\delta(q_1, 0, z) &= \{(q_1, zz), (q_2, \varepsilon)\} \\
\delta(q_1, 0, u) &= \{(q_1, zu)\} \\
\delta(q_1, 1, z) &= \{(q_1, uz)\} \\
\delta(q_1, 1, u) &= \{(q_1, uu), (q_2, \varepsilon)\} \\
\delta(q_2, 0, z) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, 1, u) &= \{(q_2, \varepsilon)\} \\
\delta(q_1, \varepsilon, z_0) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, \varepsilon, z_0) &= \{(q_2, \varepsilon)\}
\end{aligned}
$$

- Reconhecimento de 001100:

$$
\begin{aligned}
0 : (q_1, z_0) &\vdash (q_1, zz_0) \\
0 : (q_1, zz_0) &\vdash (q_1, zzz_0) \\
1 : (q_1, zzz_0) &\vdash (q_1, uzzz_0) \\
1 : (q_1, uzzz_0) &\vdash (q_2, zzz_0) \\
0 : (q_2, zzz_0) &\vdash (q_2, zz_0) \\
0 : (q_2, zz_0) &\vdash (q_2, z_0) \\
\varepsilon : (q_2, z_0) &\vdash (q_2, \varepsilon)
\end{aligned}
$$

- Esta linguagem não poderia ser reconhecida por um PDA determinístico. Por que?
- Diferente do que ocorre com os autômatos finitos, em que AFD $\equiv$ AFND, nos autômatos de pilha, PDA determinístico $\not\equiv$ PDA não $-$ determinístico;
- Isto particiona as LLC em LLC determinísticas e não-determinísticas (LLC determinísticas $\subset$ LLC não-determinísticas).

- Representação Gráfica de PDA



Convenção: $(a, w) \to z$, significando que $a \in \Sigma$ está na entrada, $w \in \Gamma$ está no topo da pilha e $z \in \Gamma^*$ será empilhado (se $z = \varepsilon$ então desempilha).

$$\delta(q_0, a, z_0) = \{(q_1, xz_0)\} \quad \delta(q_1, a, x) = \{(q_1, xx)\}$$

$$\delta(q_1, b, x) = \{(q_2, \varepsilon)\} \quad \delta(q_2, b, x) = \{(q_2, \varepsilon)\}$$

$$\delta(q_2, \varepsilon, z_0) = \{(q_0, \varepsilon)\}$$

Linguagem reconhecida:

$$L(M) = \{a^n b^n | n \geq 0\}$$

- Relação entre PDA e LLC: GLC $\Rightarrow$ PDA e PDA $\Rightarrow$ GLC;
- Teorema: Se L é uma LLC, então existe um PDA M tal que $L = N(M)$;
- Algoritmo para construir M:
  1. Colocar $G = (N, T, P, S)$ na FNG. Assumimos que $\varepsilon \notin L(G)$;
  2. $M = (\{q_1\}, T, N, \delta, q_1, S, \emptyset)$. Para cada regra $A \to a\gamma \in P$ corresponde:

$$\delta(q_1, a, A) \supset (q_1, \gamma)\,;$$

- Provar $L(G) = N(M)$.

- Exemplo: PDA para reconhecer a linguagem gerada pela seguinte gramática:

$$G = (\{S, A\}, \{a, b\}, \{S \rightarrow aAA, A \rightarrow bS|aS|a\}, S)$$

Solução:

$$M = (\{q_1\}, \{a, b\}, \{S, A\}, \delta, q_1, S, \emptyset)$$

$$\begin{aligned} \delta(q_1, a, S) &= \{(q_1, AA)\} \\ \delta(q_1, b, A) &= \{(q_1, S)\} \\ \delta(q_1, a, A) &= \{(q_1, S), (q_1, \varepsilon)\} \end{aligned}$$

- Teorema: Se L é uma linguagem reconhecida por algum PDA então L é LLC;

- Algoritmo para construir G: Seja $M = (K, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$. Então, $G = (N, \Sigma, P, S)$, onde:

  N conjunto de objetos na forma $[q, A, p]$, $q, p \in K$ e $A \in \Gamma$, unido com $\{S\}$;

  P obtido da seguinte forma:

  1. $S \rightarrow [q_0, z_0, q]$ para cada $q \in K$;
  2. $[q, A, p] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \cdots [q_m, B_m, q_{m+1}]$ para cada $q, q_1, q_2, \ldots, q_{m+1} \in K$, onde $p = q_{m+1}$, $a \in \Sigma \cup \{\varepsilon\}$, $A, B_1, B_2, \ldots, B_m \in \Gamma$, tal que $\delta(q, a, A) \supset (q_1, B_1 B_2 \cdots B_m)$. Se $m = 0$ então $q_1 = p$, $\delta(q, a, A) \supset (p, \varepsilon)$ e a produção é $[q, A, p] \rightarrow a$.

- Exemplo: Seja
  $M = (\{q_0, q_1\}, \{0, 1\}, \{X, z_0\}, \delta, q_0, z_0, \emptyset)$ e

$$\begin{aligned}
\delta(q_0, 0, z_0) &= \{(q_0, xz_0)\} \\
\delta(q_0, 0, x) &= \{(q_0, xx)\} \\
\delta(q_0, 1, x) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, 1, x) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, \varepsilon, x) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, \varepsilon, z_0) &= \{(q_1, \varepsilon)\}
\end{aligned}$$

  Construir $G = (N, T, P, S)$.

Solução:

$$N = \{S, [q_0, x, q_0], [q_0, x, q_1], [q_1, x, q_0], [q_1, x, q_1], [q_0, z_0, q_0], [q_0, z_0, q_1],$$

$$[q_1, z_0, q_0], [q_1, z_0, q_1]\}$$

$$T = \{0, 1\}$$

Produções:

$$S \to [q_0, z_0, q_0]|[q_0, z_0, q_1] \quad \text{(pela regra 1)}$$

Pela regra 2:

$$[q_0, z_0, q_0] \to 0[q_0, x, q_0][q_0, z_0, q_0]|0[q_0, x, q_1][q_1, z_0, q_0]$$

Ou seja,

$$[\overbrace{q_0}^{q}, \overbrace{z_0}^{A}, \overbrace{q_0}^{p}] \to \overbrace{0}^{a} [\overbrace{q0}^{''q_1''}, \overbrace{x}^{B_1}, \overbrace{q_0][q_0,}^{=} \overbrace{z_0}^{B_2}, \overbrace{q_0}^{q_{m+1}=p}] \quad \text{para } p = q_0$$

$$\delta(\overbrace{q_0}^{q}, \overbrace{0}^{a}, \overbrace{z_0}^{A}) = \{(\overbrace{q_0}^{''q_1''}, \overbrace{x}^{B_1} \overbrace{z_0}^{B_2=B_m})\} \quad (m = 2)$$

Observe que repetimos o lado direito da regra para $B_2 = q_0$ e $B_2 = q_1$:

$$[q_0, z_0, \overbrace{q_0}^{p=q_0}] \to 0[q_0, x, \overbrace{q_0}^{B_2=q_0}][\overbrace{q_0}^{B_2=q_0}, z_0, q_0]|0[q_0, x, \overbrace{q_1}^{B_2=q_1}][\overbrace{q_1}^{B_2=q_1}, z_0, q_0]$$

Repetindo, agora para $p = q_1$ (já que a regra exige $\forall p \in K$):

$$[q_0, z_0, \overbrace{q_1}^{p=q_1}] \to 0[q_0, x, q_0][q_0, z_0, \overbrace{q_1}^{q_{m+1}=p}]|0[q_0, x, q_1][q_1, z_0, \overbrace{q_1}^{q_{m+1}=p}]$$

Exemplificando para casos em que $m = 0$:

$$\delta(\overbrace{q_0}^{q}, \overbrace{1}^{a}, \overbrace{x}^{A}) = \{(\overbrace{q_1}^{p}, \overbrace{\varepsilon}^{m=0})\}$$

$$[\overbrace{q_0}^{q}, \overbrace{x}^{A}, \overbrace{q_1}^{p}] \rightarrow \overbrace{1}^{a} \quad (p ='' q_1'' = q_1)$$

Outro caso ($a = \varepsilon$):

$$\delta(\overbrace{q_1}^{q}, \overbrace{\varepsilon}^{a}, \overbrace{x}^{A}) = \{(\overbrace{q_1}^{p}, \overbrace{\varepsilon}^{m=0})\}$$

$$[\overbrace{q_1}^{q}, \overbrace{x}^{A}, \overbrace{q_1}^{p}] \rightarrow \overbrace{\varepsilon}^{a}$$

Solução completa:

$$
\begin{aligned}
S &\rightarrow [q_0, z_0, q_0] | [q_0, z_0, q_1] \\
[q_0, z_0, q_0] &\rightarrow 0[q_0, x, q_0][q_0, z_0, q_0] \\
&\quad\ |0[q_0, x, q_1][q_1, z_0, q_0] \\
[q_0, z_0, q_1] &\rightarrow 0[q_0, x, q_0][q_0, z_0, q_1] \\
&\quad\ |0[q_0, x, q_1][q_1, z_0, q_1] \\
[q_0, x, q_0] &\rightarrow 0[q_0, x, q_0][q_0, x, q_0] \\
&\quad\ |0[q_0, x, q_1][q_1, x, q_0] \\
[q_0, x, q_1] &\rightarrow 0[q_0, x, q_0][q_0, x, q_1] \\
&\quad\ |0[q_0, x, q_1][q_1, x, q_1] \\
&\quad\ |1 \\
[q_1, z_0, q_1] &\rightarrow \varepsilon \\
[q_1, x, q_1] &\rightarrow 1|\varepsilon
\end{aligned}
$$

Observe que não existem produções para os não-terminais $[q_1, x, q_0]$ e $[q_1, z_0, q_0]$. Assim, simplificando:

$$
\begin{aligned}
S &\rightarrow [q_0, z_0, q_1] \\
[q_0, z_0, q_1] &\rightarrow 0[q_0, x, q_1][q_1, z_0, q_1] \\
[q_0, x, q_1] &\rightarrow 0[q_0, x, q_1][q_1, x, q_1]|1 \\
[q_1, z_0, q_1] &\rightarrow \varepsilon \\
[q_1, x, q_1] &\rightarrow 1|\varepsilon
\end{aligned}
$$

- Provar $N(M) = L(G)$.

Exercícios:

1. Sejam G e G' abaixo, representadas por seu conjunto de produções:

$$\begin{aligned} P : S &\rightarrow A2D|D2A|2 \\ A &\rightarrow A1|B1|1 \\ B &\rightarrow B1|B0 \\ C &\rightarrow 0B|0 \\ D &\rightarrow 0D|0 \end{aligned}$$

$$
\begin{aligned}
P' : S &\rightarrow A|B|AB \\
A &\rightarrow aB|bS|b \\
B &\rightarrow AB|Ba|cB \\
C &\rightarrow AS|AB|b
\end{aligned}
$$

Pede-se:

(a) mostre a derivação mais à esquerda de 00211 para G e a derivação mais à direita de bb para G';

(b) monte as árvores de derivação destas duas derivações;

(c) retire todos os símbolos inúteis das duas gramáticas.

2. Seja $G = (\{S\}, \{a, b\}, \{S \rightarrow SaS|b\}, S)$.
   (a) G é ambígüa? (justifique);
   (b) caso seja ambígüa, existe alguma gramática equivalente que não seja ambígüa?

3. Seja $G = (\{A, B, C, D\}, \{a, b, c\}, P, A)$, e

$$\begin{aligned} P : A &\rightarrow ab|aBc|\varepsilon \\ B &\rightarrow CBa|Aa \\ C &\rightarrow Db|c \\ D &\rightarrow Bab|Da|\varepsilon \end{aligned}$$

Pede-se:

(a) transformar em uma gramática equivalente ε-livre;

(b) elimine todas as recursões à esquerda da gramática obtida no item anterior, se houver;

(c) fatore a gramática obtida no item anterior, se necessário.

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

$$\begin{aligned} P : S &\rightarrow Aa \\ A &\rightarrow BC \\ B &\rightarrow Sb|\varepsilon \\ C &\rightarrow cC|\varepsilon \end{aligned}$$

Pede-se:

(a) transforme G em ε-livre;

(b) elimine as produções simples;

(c) elimine as recursões à esquerda.

5. Transforme as GLC abaixo em equivalentes ε-livre e elimine as produções simples:

$$
\begin{aligned}
P : E &\rightarrow TP \\
P &\rightarrow +TP|\varepsilon \\
T &\rightarrow FQ \\
Q &\rightarrow *FQ|\varepsilon \\
F &\rightarrow (E)|id
\end{aligned}
$$

$$
\begin{aligned}
P : S &\rightarrow cSc|BA \\
A &\rightarrow aA|\varepsilon \\
B &\rightarrow bB|CA|\varepsilon \\
C &\rightarrow cCc|AS
\end{aligned}
$$

6. Elimine as recursões à esquerda:

$$\begin{aligned} P : E &\rightarrow E + T | E - T | T \\ T &\rightarrow T * F | T / F | F \\ F &\rightarrow F * * P | P \\ P &\rightarrow (E) | id \end{aligned}$$

$$\begin{aligned} P : S &\rightarrow BaS | Da | \varepsilon \\ A &\rightarrow Sa | \varepsilon \\ B &\rightarrow SAa \\ D &\rightarrow Db | b \end{aligned}$$

7. Fatore as seguintes GLC:

$$\begin{aligned} P : S &\rightarrow abC|abD \\ C &\rightarrow cC|cD|ba \\ D &\rightarrow dD|dC|ba \end{aligned}$$

$$\begin{aligned} P : S &\rightarrow bcD|Bcd \\ B &\rightarrow bB|b \\ D &\rightarrow dD|d \end{aligned}$$

8. Obtenha a FNC e a FNG:

$$\begin{aligned} P : S &\rightarrow aSb|cC \\ C &\rightarrow Dd|d \\ D &\rightarrow cC \end{aligned}$$

9. Obtenha a FNG:

$$\begin{aligned} P : S &\rightarrow AA|0 \\ A &\rightarrow SS|1 \end{aligned}$$

10. Obtenha a FNC de
$G = (\{S, T, L\}, \{a, b, +, -, *, /, [,]\}, P, S)$, onde:

$$\begin{aligned} P : S &\rightarrow T + S | T - S | T \\ T &\rightarrow L * T | L / T | L \\ L &\rightarrow [S] | a | b \end{aligned}$$

11. Construa autômatos de pilha que reconheçam as seguintes linguagens:

    (a) $L = \{w | w \in \{a, b\}^*$ tal que o número de a seja o dobro do de $b\}$;

    (b) $L = \{w | w \in \{0, 1\}^*$ tal que o número de 0 seja igual ao de $1\}$.

12. Uma estrutura de lista pode ser definida como:

    (a) λ é uma lista vazia;

    (b) a (átomo) é uma lista;

    (c) se $l_1, l_2, \ldots, l_k$ são listas, para $k \geq 1$, então $(l_1, l_2, \ldots, l_k)$ é uma lista.

    Pede-se:

    (a) Defina uma GLC que gere estruturas de listas;

    (b) Desenhe uma árvore de derivação para a sentença $((a, a), \lambda, (a))$;

    (c) Construa um PDA que reconheça listas;

    (d) Escreva os movimentos que o PDA realiza para reconhecer a sentença acima.

13. Dada a gramática $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, onde:

$$\begin{aligned} P : S &\rightarrow aAA|Bc \\ A &\rightarrow aS|BS|a \\ B &\rightarrow b|bB \end{aligned}$$

Pede-se:

(a) verifique se a sentença $abaaaa$ pertence à $L(G)$ através de derivações mais à esquerda;

(b) construa um PDA correspondente a G;

(c) mostre o reconhecimento da mesma sentença $abaaaa$ pelo PDA, através de descrição instantânea.

14. Seja o PDA $M = (\{q_0, q_1\}, \{a, b\}, \{z_0, x\}, \delta, q_0, z_0, \emptyset)$, onde:

$$\begin{aligned}
\delta(q_0, b, z_0) &= \{(q_0, xz_0)\} \\
\delta(q_0, b, x) &= \{(q_0, xx)\} \\
\delta(q_0, a, x) &= \{(q_1, x)\} \\
\delta(q_0, \varepsilon, z_0) &= \{(q_0, \varepsilon)\} \\
\delta(q_1, b, x) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, a, z_0) &= \{(q_0, z_0)\}
\end{aligned}$$

Pede-se:

(a) construa uma GLC que gera a linguagem $N(M)$;

(b) mostre a seqüência de descrições instantâneas do PDA ao reconhecer alguma sentença w de $N(M)$ com $|w| \geq 4$.

15. Seja G definida por:

$$\begin{aligned} P: S &\rightarrow 0AS|1BS|\varepsilon \\ A &\rightarrow 0AA|1 \\ B &\rightarrow 1BB|0 \end{aligned}$$

Pede-se:

(a) construa o PDA equivalente a G;

(b) mostre, através de descrições instantâneas, o reconhecimento (ou não) das seguintes sentenças:

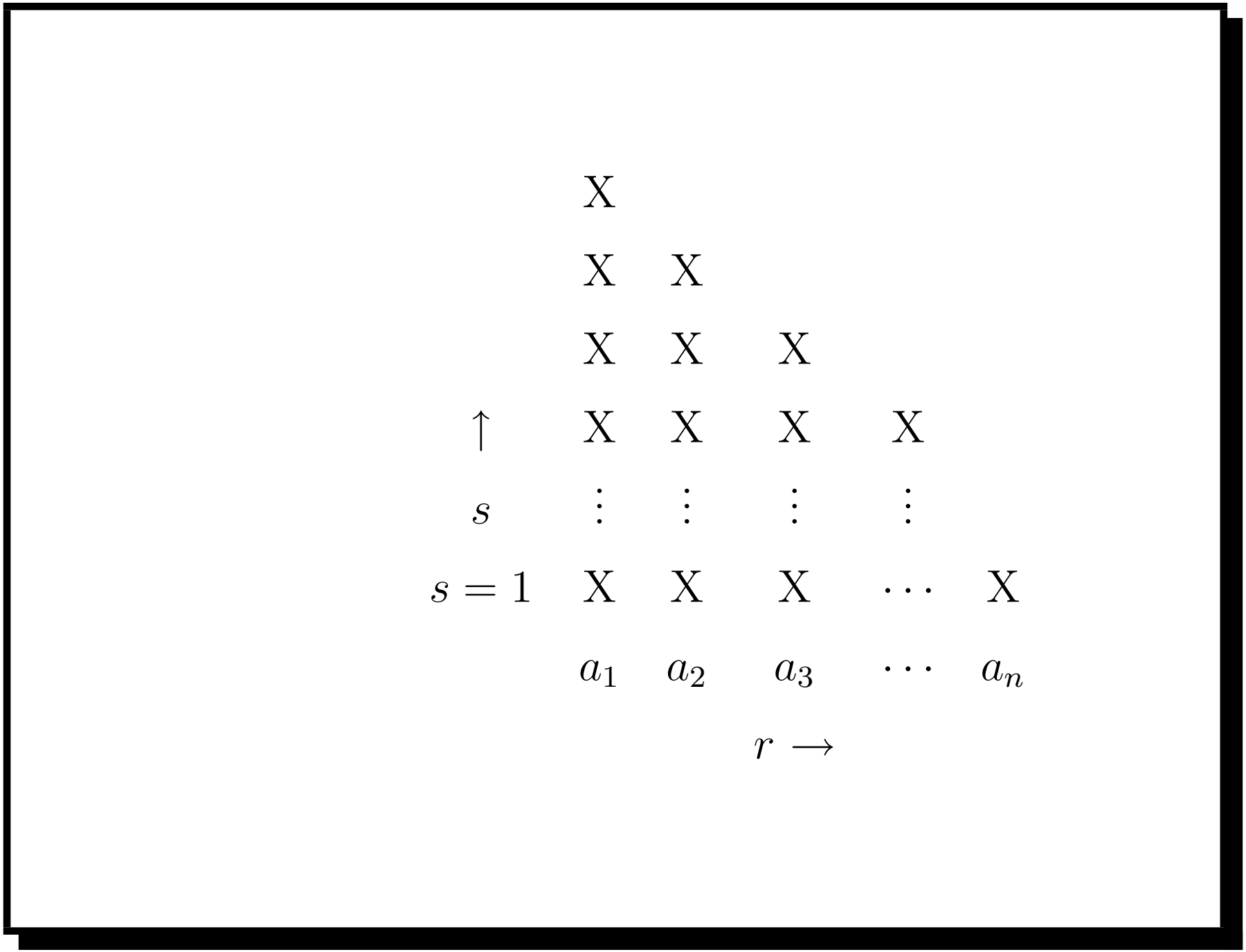
i. 001011;

ii. 111001.

4 Algoritmo CYK

- Algoritmo reconhecedor de LLC proposto por Cocke, Younger e Kasami (1965);
- Construído sobre uma GLC na FNC;
- Gera *bottom-up* todas as árvores de derivação da entrada w em tempo $O(|w|^3)$;

- Algoritmo:
  1. Suponha $G = (N, T, P, S)$, e $w = a_1 a_2 a_3 \cdots a_n$ uma entrada a ser verificada. $V(r, s)$, onde s é o número de linha e r o número de coluna, representa as células de uma matriz triangular ilustrada a seguir:


X
X X
X X X
↑ X X X X
s ⋮ ⋮ ⋮ ⋮
s = 1 X X X ··· X
a1 a2 a3 ··· an
r →

2. Variáveis que geram diretamente terminais
   $(A \rightarrow a)$:
   para r de 1 até n faça $V(r,1) = \{A | A \rightarrow a_r \in P\}$
   fim_para

3. Produção que gera duas variáveis ($A \rightarrow BC$):

   para s de 2 até n

   faça

   para r de 1 até $n - s + 1$

   faça

   $V(r, s) = \emptyset$

   para k de 1 até $s - 1$

   faça

   $V(r, s) = V(r, s) \bigcup \{A | A \rightarrow BC \in P, B \in V(r, k) \wedge$
   $C \in V(r + k, s - k)\}$

   fim_para

   fim_para

   fim_para

4. Condição de aceitação: se o símbolo inicial da gramática estiver no vértice $V(1, n)$.

Observações: O limite da iteração para r é $(n - s + 1)$ pois a tabela é triangular; Os vértices $V(r, k)$ e $V(r + k, s - k)$ são as raízes da sub-árvore de derivação de $V(r, s)$; Se uma célula for vazia significa que ela não gera nenhuma sub-árvore.

- Exemplo: $G = (\{S, A\}, \{a, b\}, P, S)$, onde $P = \{S \to AA|AS|b, A \to SA|AS|a\}$, e a entrada é $w = abaab$.

  A tabela resolvida é ilustrada a seguir:

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

Solução:

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

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 $C \in V(2,3)$.
    $V(1,4) = \emptyset \bigcup \{S, A\} = \{S, A\}$;

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.

- Exercício: Reconheça usando o algoritmo CYK.
  $P : S \to AB|BC, A \to BA|a, B \to CC|b, C \to AB|a$
  e entrada $w = baaba$.

5 Linguagens Tipo 0 e Máquinas de Turing

- As máquinas de Turing (TM) foram propostas como um modelo matemático para representar procedimentos algorítmicos;
- São um formalismo para definir *computabilidade* – conceito relacionado com os problemas que podem ser resolvidos de forma mecânica (algorítmica);
- Tese de Church: "a máquina de Turing computa exatamente o conjunto das funções que são 'computáveis' em um sentido intuitivo e informal";

- Máquinas de Turing são simultaneamente:
  - mecanismos para reconhecer *linguagens Tipo 0* ou *linguagens recursivamente enumeráveis* ou *conjuntos recursivamente enumeráveis*;
  - mecanismos para computar *funções parciais recursivas*;

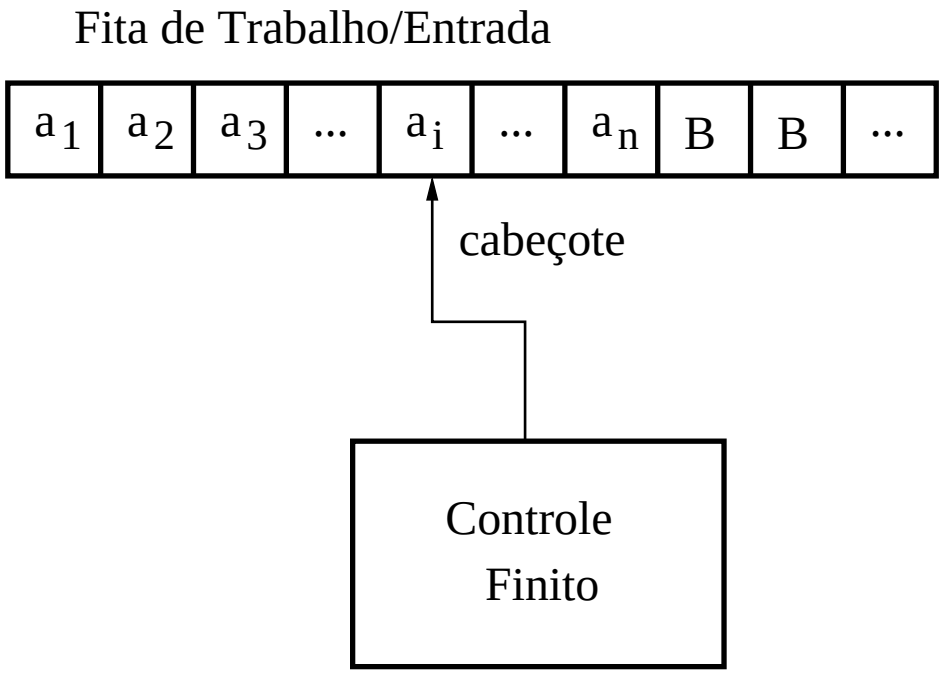
- Os conjuntos recursivamente enumeráveis são os conjuntos mais amplos que podem ser reconhecidos (ou gerados) algoritmicamente;

- As funções computadas por TM são chamadas de *funções parciais recursivas*:

  parciais a máquina pode entrar em um "laço infinito", significando que a função não está definida para um determinado valor de entrada;

  recursivas por razões históricas, são assim chamadas pois as funções computadas podem ser definidas por um formalismo recursivo;

- Modelo de Máquina de Turing



- TM é formada por:

  Fita de Trabalho/Entrada onde é inicialmente posta a sentença a ser reconhecida; formada por células que podem armazenar um símbolo; limitada à esquerda e infinita à direita;

  Cabeçote permite ler/escrever símbolos da/na fita;

  Controle Finito controla o cabeçote; representa o "programa" sendo computado pela máquina;

- Movimentos de TM:
  1. trocar de estado;
  2. gravar um símbolo na célula da fita sobre a qual está o cabeçote;
  3. redução do tamanho da string armazenada (basta escrever um "branco" sobre o símbolo do extremo direito da string armazenada);
  4. ampliação do tamanho da string armazenada (basta escrever um não-branco na posição imediatamente à direita da string);
  5. mover o cabeçote uma célula para a esquerda ou para a direita;

Observação: não podemos escrever um “branco” a não ser na última posição à direita da string armazenada (não podemos dividir a string armazenada em duas ou introduzir brancos à esquerda). Só podemos estar em uma célula com “branco” se esta for a imediatamente seguinte ao último símbolo à direita da string;

- Uma *máquina de Turing* é definida por:

$$\mathcal{M} = (K, \Sigma, \Gamma, \delta, q_0, F)$$

Onde:

K conjunto finito de estados;

$\Sigma \subset \Gamma$ conjunto finito de símbolos de entrada;

Γ conjunto finito de símbolos da fita (incluindo "branco");

$\delta : K \times \Gamma \to K \times \Gamma \times \{L, R, -\}$ função (parcial) de movimento, onde L é o movimento do cabeçote para a esquerda, R é o movimento do cabeçote para a direita, e $-$ significa que o cabeçote permanece onde está. Observe-se que só é possível escrever "branco" na fita no extremo direito da string armazenada. O movimento do cabeçote é executado *após* a escrita do símbolo na fita;

$q_0 \in K$ estado inicial;

$F \subset K$ conjunto de estados finais.

- A *configuração* de uma máquina de Turing é representada por

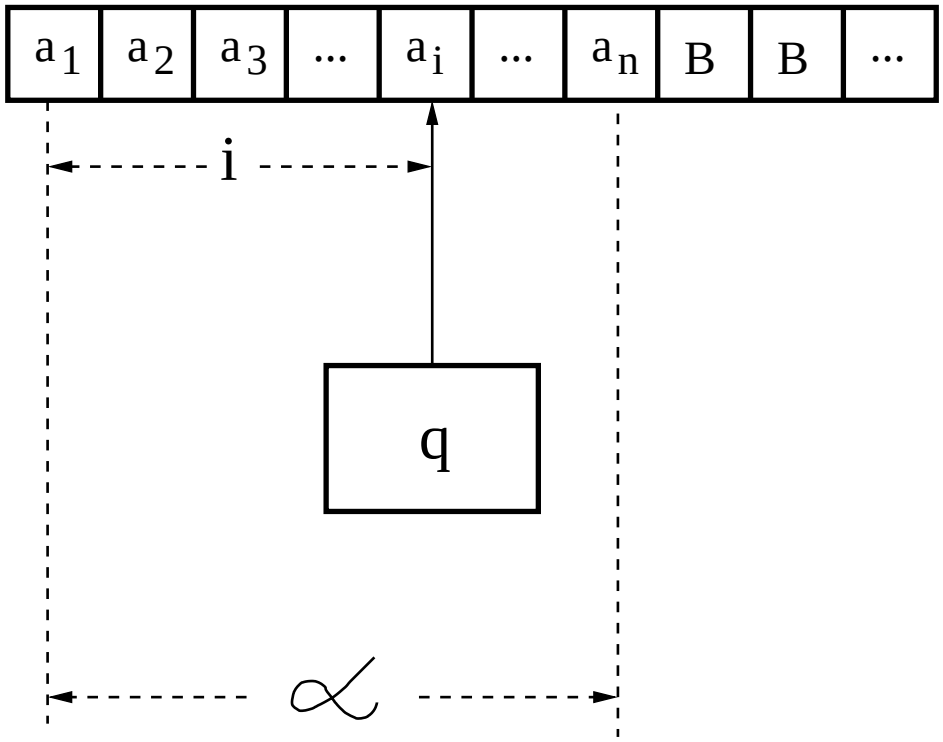
$$(q, \alpha, i)$$

Onde:

$q \in K$ estado atual da máquina;

$\alpha \in (\Gamma - \{B\})^*$ string armazenada na fita;

i posição do cabeçote.


a1
a2
a3
...
ai
...
an
B
B
...
i
q
α

- A *descrição instantânea* de TM é definida como:

$$\alpha_1 q \alpha_2$$

onde:

$q \in K$ estado corrente da máquina;

$\alpha_1, \alpha_2 \in \Gamma^*$ seqüências de símbolos representando os lados direito e esquerdo do cabeçote.

Observação: assume-se que o cabeçote está apontando para o símbolo mais à esquerda de α_2. Se $\alpha_2 = \varepsilon$ então o cabeçote está apontando para B. Se $\alpha_1 = \varepsilon$ então o cabeçote está na primeira posição da string armazenada ($i = 1$).

- Movimentos de TM:
  - Se $1 \leq i \leq n$, $x \in (\Gamma - \{B\})$ e $\delta(q, a_i) = (p, x, R)$ então $(q, a_1 a_2 \cdots a_i \cdots a_n, i) \vdash$ $(p, a_1 a_2 \cdots a_{i-1} x a_{i+1} \cdots a_n, i+1)$;
  - Se $2 \leq i \leq n$, $x \in (\Gamma - \{B\})$ e $\delta(q, a_i) = (p, x, L)$ então $(q, a_1 a_2 \cdots a_i \cdots a_n, i) \vdash$ $(p, a_1 a_2 \cdots a_{i-1} x a_{i+1} \cdots a_n, i-1)$;
  - Se $1 \leq i \leq n$, $x \in (\Gamma - \{B\})$ e $\delta(q, a_i) = (p, x, -)$ então $(q, a_1 a_2 \cdots a_i \cdots a_n, i) \vdash (p, a_1 a_2 \cdots a_{i-1} x a_{i+1} \cdots a_n, i)$

- Se $x \in (\Gamma - \{B\})$ e $\delta(q, B) = (p, x, -)$ então
  $(q, a_1 a_2 \cdots a_n, n+1) \vdash (p, a_1 a_2 \cdots a_n x, n+1)$;
- Se $x \in (\Gamma - \{B\})$ e $\delta(q, B) = (p, x, L)$ então
  $(q, a_1 a_2 \cdots a_n, n+1) \vdash (p, a_1 a_2 \cdots a_n x, n)$;
- Se $x \in (\Gamma - \{B\})$ e $\delta(q, B) = (p, x, R)$ então
  $(q, a_1 a_2 \cdots a_n, n+1) \vdash (p, a_1 a_2 \cdots a_n x, n+2)$;
- Se $\delta(q, a_n) = (p, B, -)$ então
  $(q, a_1 a_2 \cdots a_n, n) \vdash (p, a_1 a_2 \cdots a_{n-1}, n)$;
- Se $\delta(q, a_n) = (p, B, L)$ então
  $(q, a_1 a_2 \cdots a_n, n) \vdash (p, a_1 a_2 \cdots a_{n-1}, n-1)$;

- Se $\delta(q, B) = (p, B, -)$ então
  $(q, a_1 \cdots a_n, n+1) \vdash (p, a_1 \cdots a_n, n+1)$;
- Se $\delta(q, B) = (p, B, L)$ então
  $(q, a_1 \cdots a_n, n+1) \vdash (p, a_1 \cdots a_n, n)$.

- Linguagem aceita por TM

  Seja $\mathcal{M} = (K, \Sigma, \Gamma, \delta, q_0, F)$, então:
  $L(\mathcal{M}) = \{w | w \in \Sigma^* \wedge \exists q \in F, \alpha \in \Gamma^*, i \geq 0 (q_0, w, 1) \vdash (q, \alpha, i)\}$.

- Parada de TM – se uma TM aceita uma linguagem L então ela pára para todas as palavras de L;

  Existem duas possibilidades de não reconhecer uma palavra:

  1. o processo de reconhecimento trancar antes de ser encontrado um estado final;
  2. a palavra não é reconhecida pois a máquina não pára (função parcial).

- Relação entre TM e Gramática Tipo 0
- Teorema: Se L é gerada por uma gramática Tipo 0 então L pode ser reconhecida por uma máquina de Turing;
- Teorema: Se L é reconhecida por uma máquina de Turing então L é gerada por uma gramática Tipo 0.

- Exemplo: $L = \{0^n 1^n | n \geq 1\}$. Solução:

$$\mathcal{M} = (K, \Sigma, \Gamma, \delta, q_0, F)$$

$$K = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{0, 1\}$$

$$\Gamma = \{0, 1, x, y, B\}$$

$$F = \{q_5\}$$

$$\delta(q_0, 0) = (q_1, x, R) \quad \delta(q_1, 0) = (q_1, 0, R)$$

$$\delta(q_1, 1) = (q_2, y, L) \quad \delta(q_2, y) = (q_2, y, L)$$

$$\delta(q_2, x) = (q_3, x, R) \quad \delta(q_2, 0) = (q_4, 0, L)$$

$$\delta(q_4, 0) = (q_4, 0, L) \quad \delta(q_4, x) = (q_0, x, R)$$

$$\delta(q_3, y) = (q_3, y, R) \quad \delta(q_3, B) = (q_5, B, -)$$

$$\delta(q_1, y) = (q_1, y, R)$$

Reconhecimento de 0011:

$$
\begin{aligned}
(q_0, 0011, 1) &\vdash (q_1, x011, 2) \vdash \\
&\vdash (q_1, x011, 3) \vdash \\
&\vdash (q_2, x0y1, 2) \vdash \\
&\vdash (q_4, x0y1, 1) \vdash \\
&\vdash (q_0, x0y1, 2) \vdash \\
&\vdash (q_1, xxy1, 3) \vdash \\
&\vdash (q_1, xxy1, 4) \vdash \\
&\vdash (q_2, xxyy, 3) \vdash \\
&\vdash (q_2, xxyy, 2) \vdash \\
&\vdash (q_3, xxyy, 3) \vdash
\end{aligned}
$$

$$\begin{aligned} &\vdash \quad (q_3, xxyy, 4) \vdash \\ &\vdash \quad (q_3, xxyy, 5) \vdash \\ &\vdash \quad (q_5, xxyy, 5) \end{aligned}$$

- Função computada por TM – podemos considerar a máquina de Turing também como um mecanismo para computar funções parciais recursivas (funções sobre strings);
- Se considerarmos a tripla (q_0, δ, F) como sendo um programa p, então podemos considerar a função $\mathcal{M}_p$ como sendo a função computada pelo programa p na máquina $\mathcal{M}$.

- Existem diversas variações da definição de máquina de Turing:
  1. TM não-determinística;
  2. fita infinita em ambas as direções;
  3. TM com mais de um cabeçote;
  4. TM com mais de uma fita;
  5. combinações destas possibilidades;
- Todas são equivalentes à definição original (TM original *simula* todas as suas variações);
- Isto é uma evidência da correção da tese de Church.

Exercícios:

1. Defina uma TM para reconhecer
   $L = \{w | w \in \{0,1\}^* \wedge w = w^R\}$ (palíndromes);
2. Defina uma TM para reconhecer $L = \{a^n b^n c^n | n \geq 0\}$
   e mostre o reconhecimento de $aabbcc$.

6 LSC e Autômatos de Fita Limitada

- Um *autômato de fita limitada* – Linear Bounded Automata (LBA) – é uma máquina de Turing *não-determinística* em que o cabeçote nunca abandona aquelas células sobre as quais a sentença de entrada foi gravada;
- O LBA é o reconhecedor das linguagens sensíveis ao contexto (LSC);

- Definição formal de LBA

$$M = (K, \Sigma, \Gamma, \delta, q_0, F)$$

Onde:

K conjunto finito de estados;

$\Sigma \subset \Gamma$ conjunto finito de símbolos de entrada;

Γ conjunto finito de símbolos da fita;

$\delta : K \times \Gamma \rightarrow \text{Partes}(K \times \Gamma \times \{L, R, -\})$;

$q_0 \in K$ estado inicial;

$F \subset K$ conjunto finito de estados finais.

Observação: o conjunto Γ possui dois símbolos especiais $\pounds$ e \$ que são os limites à esquerda e à direita da sentença de entrada.

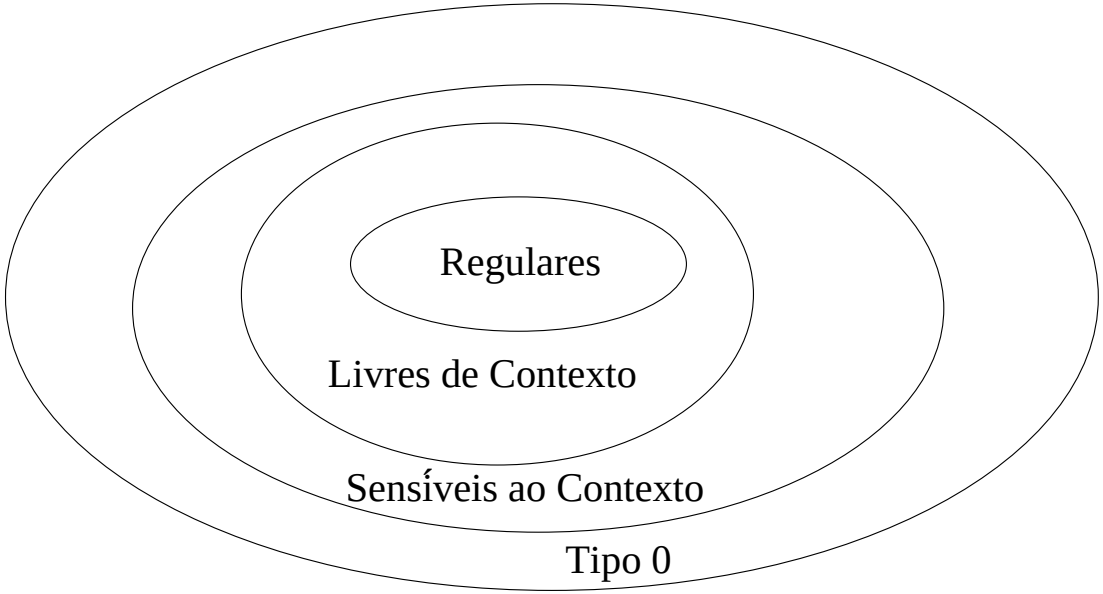
- Linguagem aceita por um LBA:
  $L = \{w | w \in \Sigma^* \wedge \exists q \in F, \alpha \in \Gamma^*, i \geq 0 (q_0, \pounds w\$, 1) \vdash (q, \alpha, i)\}$.

  Teorema: Se L é uma LSC então L é aceita por algum LBA.

  Teorema: Se L é reconhecida por um LBA então $L - \{\varepsilon\}$ é uma LSC.

7 Hierarquia de Chomsky (revisitada e ampliada)

Relembrando:

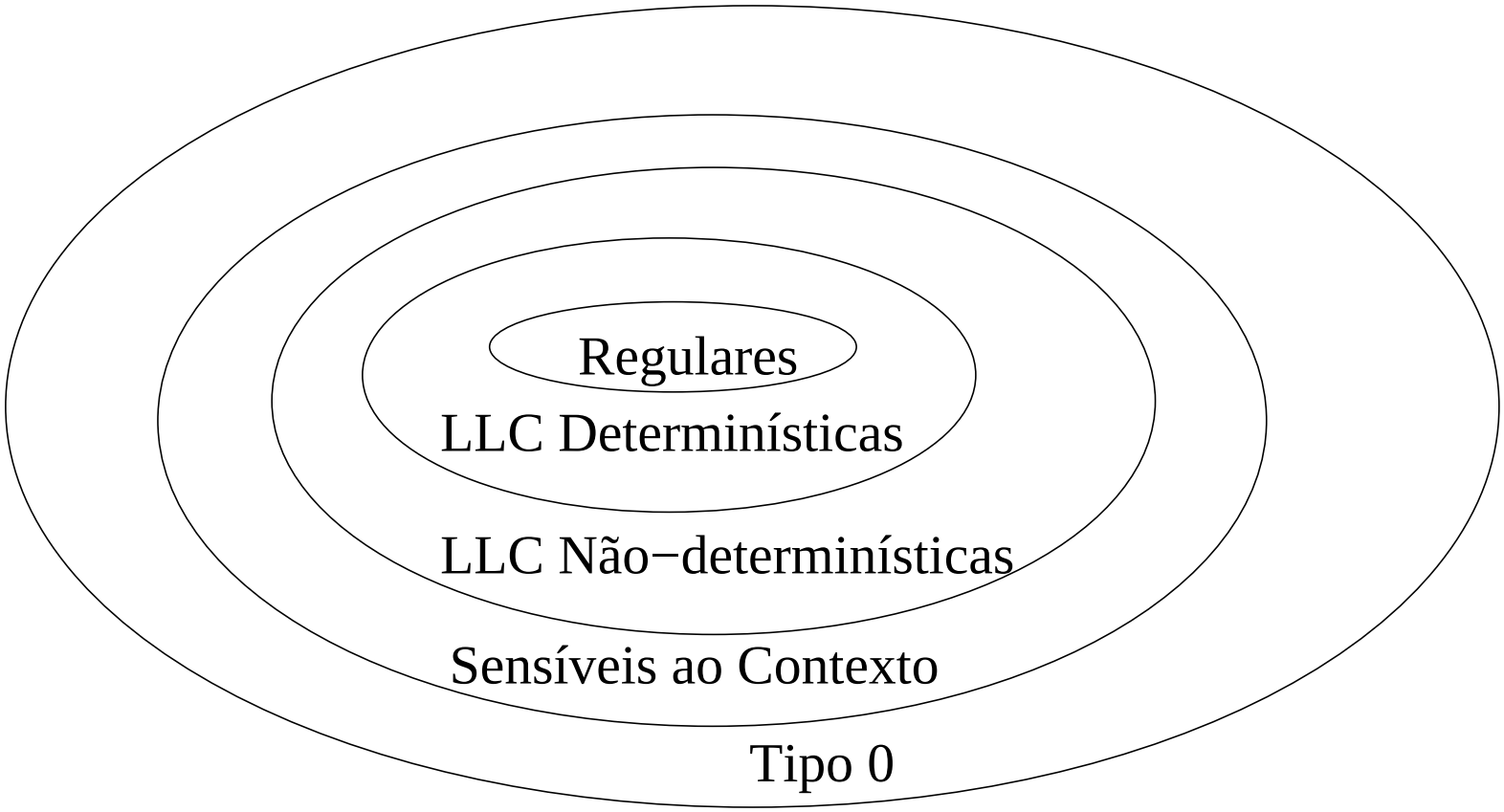


$$\text{Tipo } 3 \subset \text{Tipo } 2 \subset \text{Tipo } 1 \subset \text{Tipo } 0 \subset T^*$$

- Nos extremos da hierarquia (AF e TM – linguagens regulares e Tipo 0) existe equivalência entre determinismo e não-determinismo (mesma classe de linguagens reconhecidas);
- Para o autômato de fita limitada não se sabe se determinismo e não-determinismo aceitam a mesma classe de linguagens ou classes diferentes (questão em aberto; não existe prova matemática);

- Para o PDA sabemos que não existe equivalência entre determinismo e não-determinismo;
- Linguagens livres de contexto determinísticas (LLCD) e linguagens livres de contexto não-determinísticas (LLCND);
- As LLCD são a classe de linguagens que pode ser reconhecida por um compilador de forma eficiente (em tempo polinomial);
- São a classe mais importante pois representam a sintaxe da maioria das linguagens de programação;
- As LLCD são geradas pelas *gramáticas LR*;

Teorema: Se L é uma LLC e o complemento de L $(T^* - L)$ não é, então L não é LLCD.



$$\text{Tipo } 3 \subset \text{LLCD} \subset \text{LLCND} \subset \text{Tipo } 1 \subset \text{Tipo } 0 \subset T^*$$

- É possivel caracterizar toda a hierarquia de Chomsky ampliada apresentada até aqui usando-se apenas o autômato de pilha;
- Definimos PDA(n) como sendo um autômato de pilha que possui n pilhas. Então:
  1. PDA(0) é equivalente ao AF;
  2. PDA(1) = PDA caracterizando as LLCD e LLCND dependendo da existência ou não de não-determinismo;
  3. PDA(n) $\equiv$ TM, para $n \geq 2$ (por que?). Logo, PDA(n) $\equiv$ PDA(2), para todo $n > 2$ – adicionar mais pilhas não muda nada (por que?).

FIM