Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

Departamento de Informática Teórica

Instituto de Informática / UFRGS

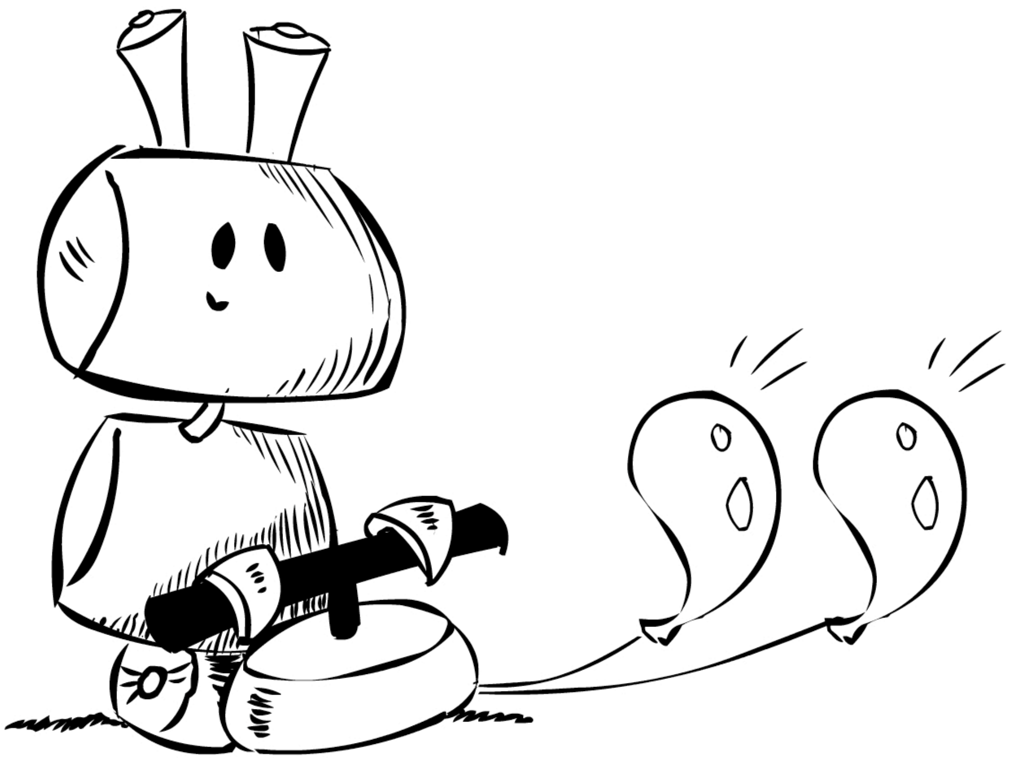
Linguagens Formais e Autômatos

P. Blauth Menezes

6 – Propriedades e Reconhecimento das LLC

7.1 Propriedades das LLC
- 7.1.1 Investigação se é LLC
- 7.1.2 Operações Fechadas sobre as LLC
- 7.1.3 Investigação se é Vazia, Finita ou Infinita

7.2 Algoritmos de Reconhecimento
- 7.2.1 Autômato com Pilha como Reconhecedor
- 7.2.2 Algoritmo de Cocke-Younger-Kasami
- 7.2.3 Algoritmo de Early



7 – Propriedades e Reconhecimento das LLC

7 Propriedades e Reconhecimento das LLC

◆ A Classe das LLC

- mais geral que a das Regulares
- mas ainda é relativamente restrita

◆ Fácil definir linguagens que não são LLC

- *Palavra duplicada*
  - { ww | w é palavra sobre { a, b } }
  - analogia com questões similares de linguagens de programação
  - exemplo: declaração de uma variável antes do seu uso
- *Triplo balanceamento*
  - $\{ a^n b^n c^n \mid n \geq 0 \}$
  - importante limitação das LLC
  - (propositalmente) incomum nas linguagens de programação
  - **`se-então-senão`** é uma estrutura deste tipo? Exercício
- usando Autômato com Pilha
  - fácil intuir por que *não* são LLC

◆ Assim

- como determinar se uma linguagem é LLC?

- é fechada para operações
  - ∗ união?
  - ∗ intersecção?
  - ∗ concatenação?
  - ∗ complemento?

- como verificar se uma linguagem livre do contexto é
  - ∗ infinita
  - ∗ finita (ou até mesmo vazia)?

- **Algumas questões *não* possuem solução computacional**
  - *não* existe algoritmo capaz de
    - analisar duas LLC quaisquer
    - concluir se são iguais ou diferentes
- **Uma palavra pertence ou não a uma linguagem ?**
  - uma das principais questões relacionadas com LF
- **Algoritmos de reconhecimento**
  - válidos para qualquer linguagem da classe
  - importante: quantidade de recursos que o algoritmo necessita
    - tempo & espaço

◆ Algoritmos construídos a partir de uma GLC

- Autômato com Pilha Descendente
- Algoritmo de Cocke-Younger-Kasami
- Algoritmo de Early

◆ Reconhecedores que usam autômato com pilha

- muito simples
- em geral, ineficientes
- tempo de processamento é proporcional a $k^{|w|}$
  - w palavra de entrada
  - k depende do autômato

◆ Existem algoritmos mais eficientes

- não baseados em AP
  - tempo de processamento proporcional a $|w|^3$
  - até menos
- tempo proporcional a $|w|^3$: não é provado
  - se é efetivamente necessário
  - para que um algoritmo genérico reconheça LLC

6 – Propriedades e Reconhecimento das LLC

7.1 Propriedades das LLC
- 7.1.1 Investigação se é LLC
- 7.1.2 Operações Fechadas sobre as LLC
- 7.1.3 Investigação se é Vazia, Finita ou Infinita

7.2 Algoritmos de Reconhecimento
- 7.2.1 Autômato com Pilha como Reconhecedor
- 7.2.2 Algoritmo de Cocke-Younger-Kasami
- 7.2.3 Algoritmo de Early

7.1 Propriedades das LLC

7.1.1 Investigação se é LLC

- **Para mostrar que é LLC**
  - expressá-la usando os formalismos livre do contexto
    - Gramática Livre do Contexto
    - Autômato com Pilha

- **Demonstração de que não é LLC**
  - realizada caso a caso
  - Lema do Bombeamento para as LLC
    - útil no estudo das propriedades
    - pode ser usado para verificar se não é LLC

Teorema: Bombeamento para as LLC

Se L é uma LLC, então

- existe uma constante n tal que
- para qualquer palavra $w \in L$ onde $|w| \geq n$,
- w pode ser definida como $w = u\,x\,v\,y\,z$
  - ∗ $|x\,v\,y| \leq n$,
  - ∗ $|x\,y| \geq 1$
- para todo $i \geq 0$, $u\,x^i\,v\,y^i\,z$ é palavra de L

Prova:

Uma maneira é usando gramáticas na Forma Normal de Chomsky

- se a gramática possui s variáveis
- pode-se assumir que $n = 2^s$
- prova não será detalhada

- **Para $w = u\,x\,v\,y\,z$ tal que $|xy| \geq 1$**
  - x ou y pode ser a palavra vazia (mas não ambas)
  - resulta em bombeamentos não balanceados
    - linguagens regulares
  - duplo bombeamento balanceado
    - importante característica das LLC
  - "duplo balanceamento": $\{ a^n b^n \mid n \geq 0 \}$
  - "palavra e sua reversa": $\{ ww^r \mid w$ pertence a $\{ a, b \}^* \}$

◆ Conclusões a partir dos lemas do bombeamento

- regulares são capazes de expressar
  - apenas bombeamentos
  - sem qualquer balanceamento
- livres do contexto são capazes de expressar
  - bombeamentos balanceados
  - dois a dois
- livres do contexto *não* são capazes de expressar
  - triplo bombeamento balanceado

Exp: Triplo Balanceamento - $L = \{ a^n b^n c^n \mid n \geq 0 \}$

Prova por absurdo. Suponha que L é LLC

Então existe uma gramática na Forma Normal de Chomsky G

- com s variáveis
- gera palavras não-vazias de L
- sejam $r = 2^s$ e $w = a^r b^r c^r$

Pelo bombeamento, w pode ser definida como $w = u x v y z$

- $|x v y| \leq r$
- $|x y| \geq 1$
- para todo $i \geq 0$, $u x^i v y^i z$ é palavra de L

Absurdo!!!.

- Por que?

De fato, como $|x\,v\,y| \leq r$

- não é possível supor que $x\,y$ possui símbolos a e c
- quaisquer ocorrências de a e c
  - estão separadas por, pelo menos, r ocorrências de b
- $x\,y$ jamais possuirá ocorrências de a, b e c simultaneamente
  - aplicação do bombeamento
  - pode desbalancear as ocorrências de a, b e c

6 – Propriedades e Reconhecimento das LLC

7.1 Propriedades das LLC
- 7.1.1 Investigação se é LLC
- 7.1.2 Operações Fechadas sobre as LLC
- 7.1.3 Investigação se é Vazia, Finita ou Infinita

7.2 Algoritmos de Reconhecimento
- 7.2.1 Autômato com Pilha como Reconhecedor
- 7.2.2 Algoritmo de Cocke-Younger-Kasami
- 7.2.3 Algoritmo de Early

7.1.2 Operações Fechadas sobre as LLC

◆ Operações sobre linguagens podem ser usadas para

- construir novas linguagens
  - a partir de linguagens conhecidas
  - definindo-se uma álgebra
- provar propriedades
- construir algoritmos

◆ Classe das LLC

- União ✔
- concatenação ✔
- intersecção ✘
- complemento ✘

Teorema: Operações Fechadas sobre LLC

- União
- Concatenação

Prova:

União: (*direta*) AP + não-determinismo (GLC: exercício)

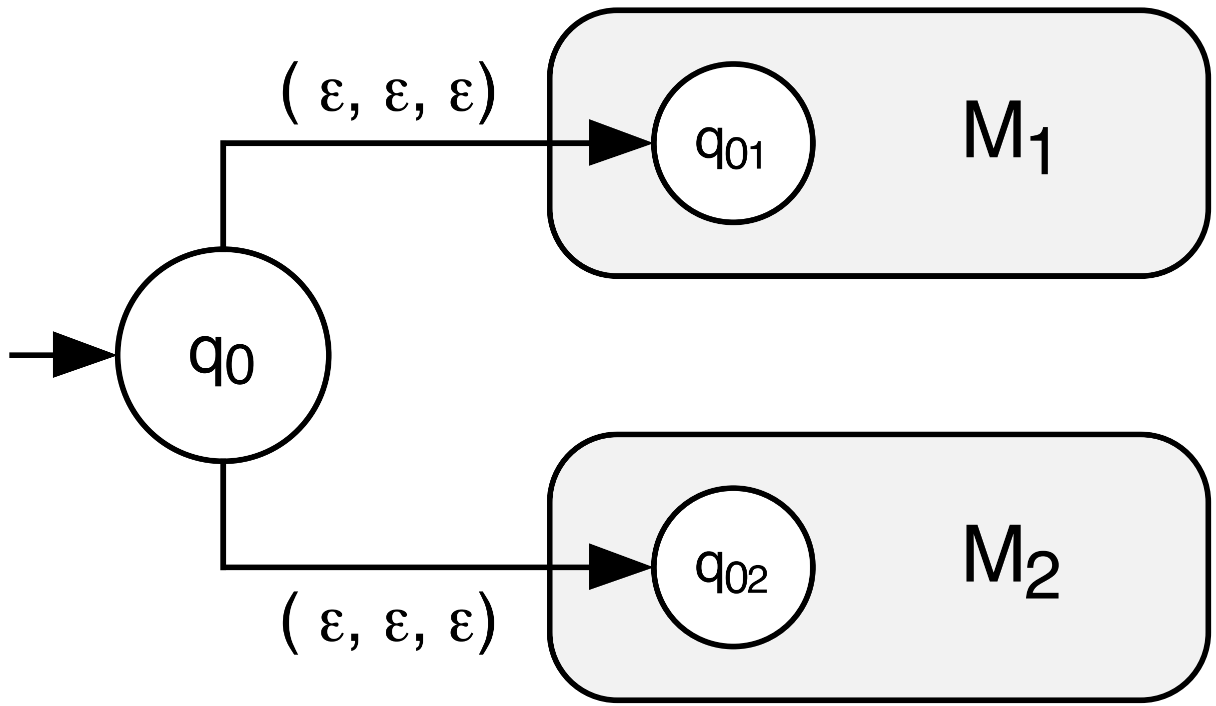
Suponha L_1 e L_2, LLC. Então, existem AP

$$M_1 = (\Sigma_1, Q_1, \delta_1, q0_1, F_1, V_1) \quad e \quad M_2 = (\Sigma_2, Q_2, \delta_2, q0_2, F_2, V_2)$$

tais que $ACEITA(M_1) = L_1$ e $ACEITA(M_2) = L_2$

Seja M_3 (suponha que $Q_1 \cap Q_2 \cap \{ q_0 \} = \varnothing$ e $V_1 \cap V_2 = \varnothing$)

$$M_3 = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{ q_0 \}, \delta_3, q_0, F_1 \cup F_2, V_1 \cup V_2)$$



Claramente, M_3 reconhece $L_1 \cup L_2$

Concatenação: (*direta*) GLC (AP: exercício)

Suponha L_1 e L_2, LLC. Então, existem GLC

$$G_1 = (V_1, T_1, P_1, S_1) \quad e \quad G_2 = (V_2, T_2, P_2, S_2)$$

tais que $GERA(G_1) = L_1$ e $GERA(G_2) = L_2$

Seja G_3 (suponha que $V_1 \cap V_2 \cap \{ S \} = \varnothing$)

$$G_3 = (V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup \{ S \rightarrow S_1 S_2 \}, S)$$

Claramente qualquer palavra gerada por G_3 terá, como

- prefixo, uma palavra de L_1
- sufixo, uma palavra de L_2

Logo, $L_1 L_2$ é LLC

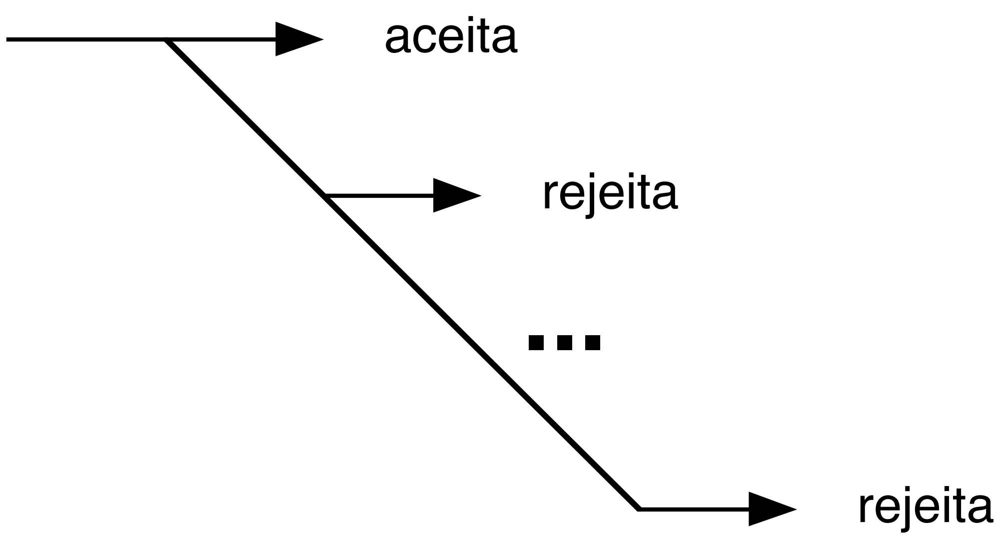
- **Teorema a seguir mostra que a Classe das LLC não é fechada para**
  - intersecção
  - complemento
- **Aparentemente, uma contradição**
  - foi verificado que, se L é LLC
    - existe AP M tal que $\text{ACEITA}(M) = L$ e $\text{REJEITA}(M) = \sim L$
    - rejeita qualquer palavra que não pertença a L
  - teorema a seguir
    - se L é LLC
    - *não* se pode afirmar que $\sim L$ também é LLC

◆ Assim

- é possível um AP *rejeitar* o complemento de uma LLC
- embora nem sempre seja possível *aceitar* o complemento

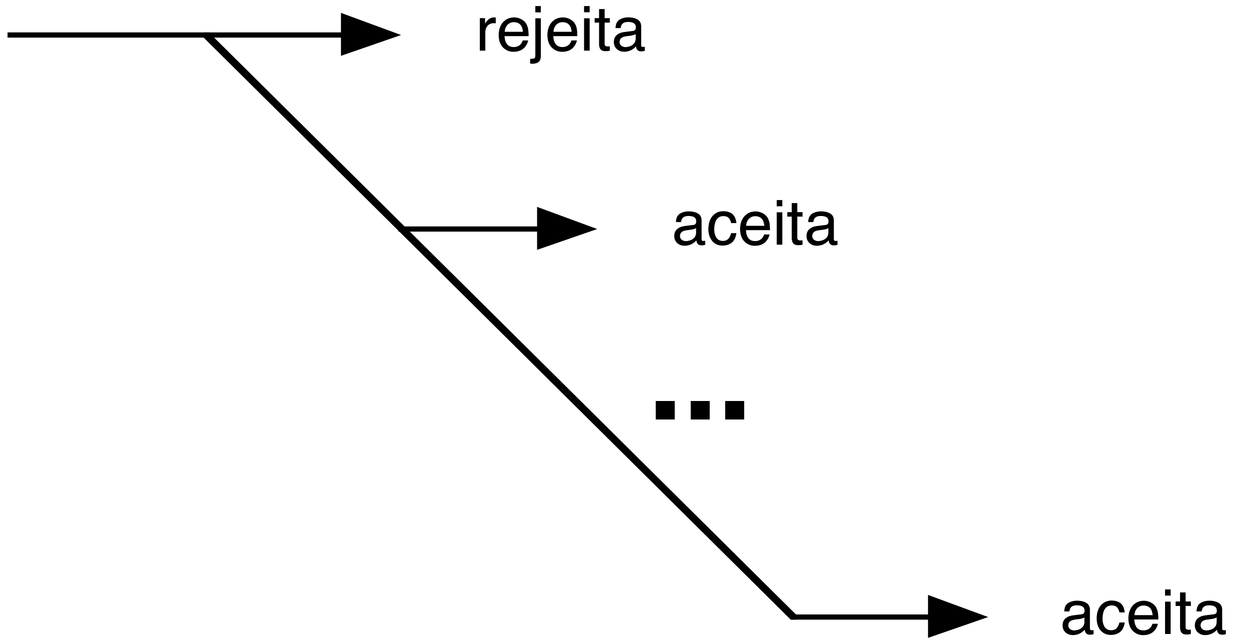
◆ Explicação intuitiva: AP

- aceita se pelo menos um dos caminhos alternativos aceita



◆ Inversão de aceita/rejeita

- situação continua sendo de aceitação



Teorema: Operações Não-Fechadas sobre Linguagens Livres do Contexto

- Intersecção
- Complemento

Prova:

Intersecção: (*direta*) contra-exemplo

Sejam LLC

$$L_1 = \{ a^n b^n c^m \mid n \geq 0 \text{ e } m \geq 0 \} \quad \text{e} \quad L_2 = \{ a^m b^n c^n \mid n \geq 0 \text{ e } m \geq 0 \}$$

Intersecção de L_1 com L_2, não é LLC

$$L_3 = \{ a^n b^n c^n \mid n \geq 0 \}$$

Complemento: (*direta*) Conseqüência direta (por quê?)

Intersecção pode ser representada em termos da união e do complemento

Classe das LLC não é fechada para a operação de intersecção

- como é fechada para a união
- não se pode afirmar que é fechada para o complemento

6 – Propriedades e Reconhecimento das LLC

- 7.1 Propriedades das LLC
  - 7.1.1 Investigação se é LLC
  - 7.1.2 Operações Fechadas sobre as LLC
  - 7.1.3 Investigação se é Vazia, Finita ou Infinita
- 7.2 Algoritmos de Reconhecimento
  - 7.2.1 Autômato com Pilha como Reconhecedor
  - 7.2.2 Algoritmo de Cocke-Younger-Kasami
  - 7.2.3 Algoritmo de Early

7.1.3 Investigação se é Vazia, Finita ou Infinita

Teorema: LLC Vazia, Finita ou Infinita

Prova: Suponha L LLC

Vazia: (*direta*)

Seja $G = (V, T, P, S)$, GLC tal que $GERA(G) = L$

Seja $G_{SI} = (V_{SI}, T_{SI}, P_{SI}, S)$

- equivalente a G
- excluindo os símbolos inúteis

Se P_{SI} for vazio, então L é vazia

Finita e Infinita: (*direta*)

Seja $G = (V, T, P, S)$ GLC tal que $GERA(G) = L$

Seja $G_{FNC} = (V_{FNC}, T_{FNC}, P_{FNC}, S)$ equivalente na FN de Chomsky

Se existe A variável tal que

- $A \rightarrow BC$ — A no lado esquerdo
- $X \rightarrow YA$ ou $X \rightarrow AY$ — A no lado direito
- existe um ciclo em A

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

- A é capaz de gerar palavras de qualquer tamanho
- L é infinita

Caso não exista tal A, então L é finita

6 – Propriedades e Reconhecimento das LLC

7.2 Algoritmos de Reconhecimento

- **Podem ser classificados como**
  - *Top-Down* ou *Preditivo*
    - construí uma árvore de derivação
    - a partir da raiz (símbolo inicial da gramática)
    - com ramos em direção às folhas (terminais)
  - *Bottom-Up*
    - oposto do *top-down*
    - parte das folhas
    - construindo a árvore de derivação em direção à raiz

6 – Propriedades e Reconhecimento das LLC

7.2.1 Autômato com Pilha como Reconhecedor

◆ **Reconhecedores usando AP**

- construção relativamente simples e imediata
- relação quase direta entre produções e as transições do AP
- algoritmos
  - *top-down*
  - simula derivação mais à esquerda
  - não-determinismo: produções alternativas da gramática

◆ Foi visto que qq LLC pode ser especificada por um AP

- a partir de uma Gramática na Forma Normal de Greibach
  - ∗ cada produção gera exatamente um terminal
  - ∗ geração de w: $|w|$ etapas de derivação
- cada variável: pode ter diversas produções associadas
  - ∗ AP testa as diversas alternativas
  - ∗ número de passos para reconhecer w: proporcional a $k^{|w|}$
  - ∗ aproximação de k: metade da média 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 à esquerda
  - simula a derivação mais à esquerda

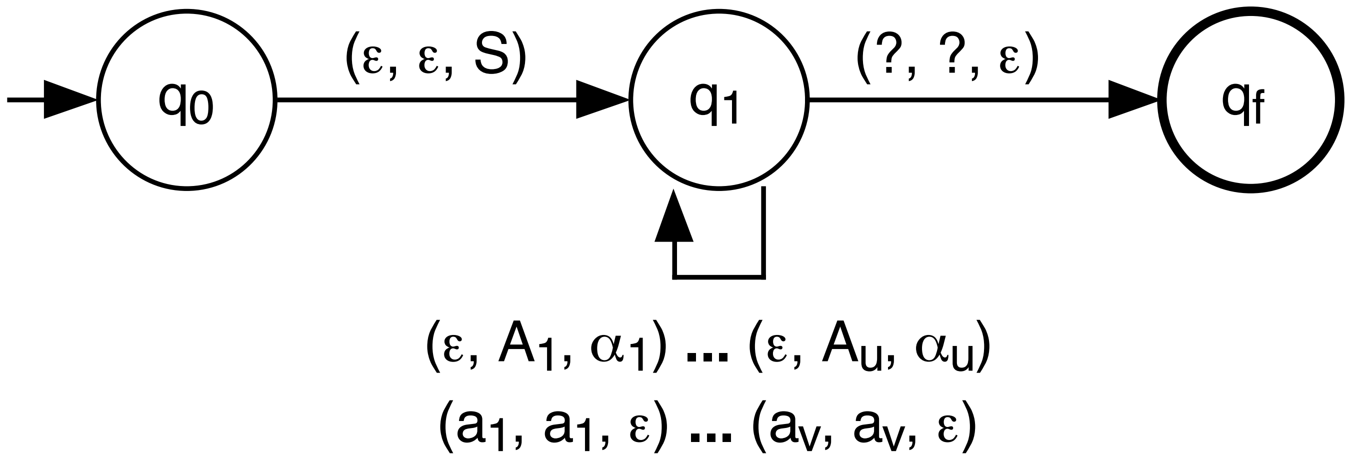
- algoritmo
  - inicialmente, empilha o símbolo inicial
  - topo = variável: substitui, (não-determinismo), por todas as produções da variável
  - topo = terminal: testa se é igual ao próximo símbolo da entrada

Def: Algoritmo: Autômato com Pilha Descendente

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

$$M = (T, \{ q_0, q_1, q_f \}, \delta, q_0, \{ q_f \}, V \cup T)$$

- $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_1, S) \}$
- $\delta(q_1, \varepsilon, A) = \{ (q_1, \alpha) \mid A \rightarrow \alpha \in P \}$ — A de V
- $\delta(q_1, a, a) = \{ (q_1, \varepsilon) \}$ — a de T
- $\delta(q_1, ?, ?) = \{ (q_f, \varepsilon) \}$



Exp: AP Descendente: Duplo Balanceamento

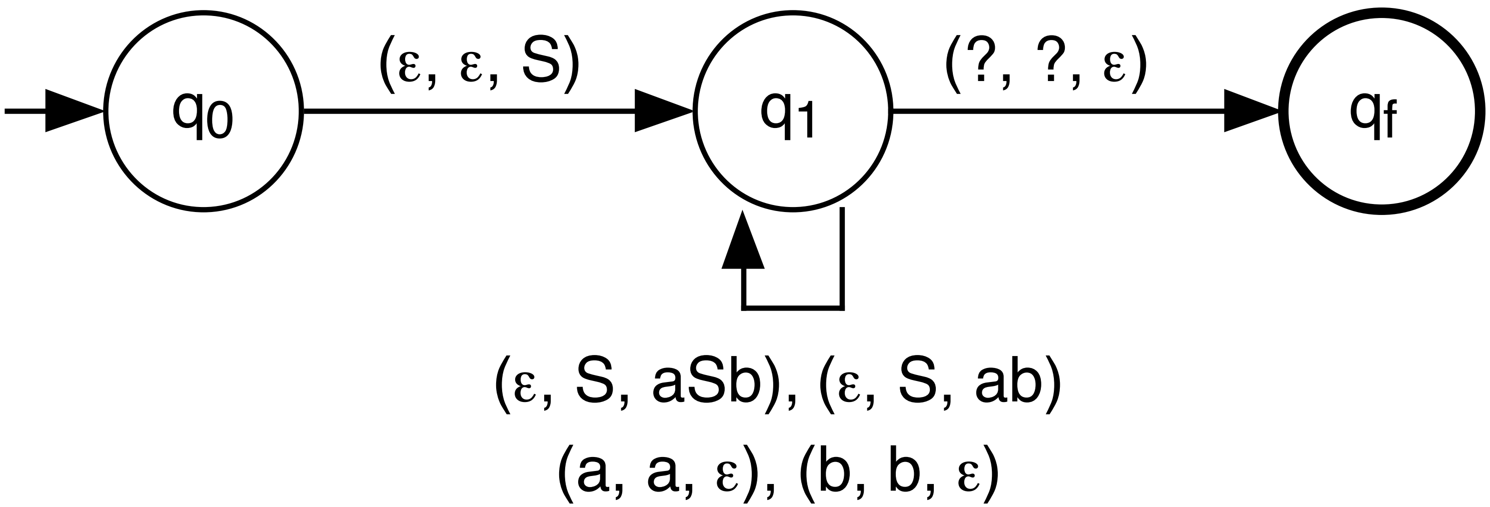
$L = \{ a^n b^n \mid n \geq 1 \}$

- $G = (\{ S \}, \{ a, b \}, P, S)$
- $P = \{ S \to aSb \mid ab \}$

GLC sem recursão à esquerda

Autômato com pilha descendente

$M = (\{ a, b \}, \{ q_0, q_1, q_f \}, \delta, q_0, \{ q_f \}, \{ S, a, b \})$



6 – Propriedades e Reconhecimento das LLC

7.1.2 Algoritmo de Cocke-Younger-Kasami

◆ Algoritmo de Cocke-Younger-Kasami

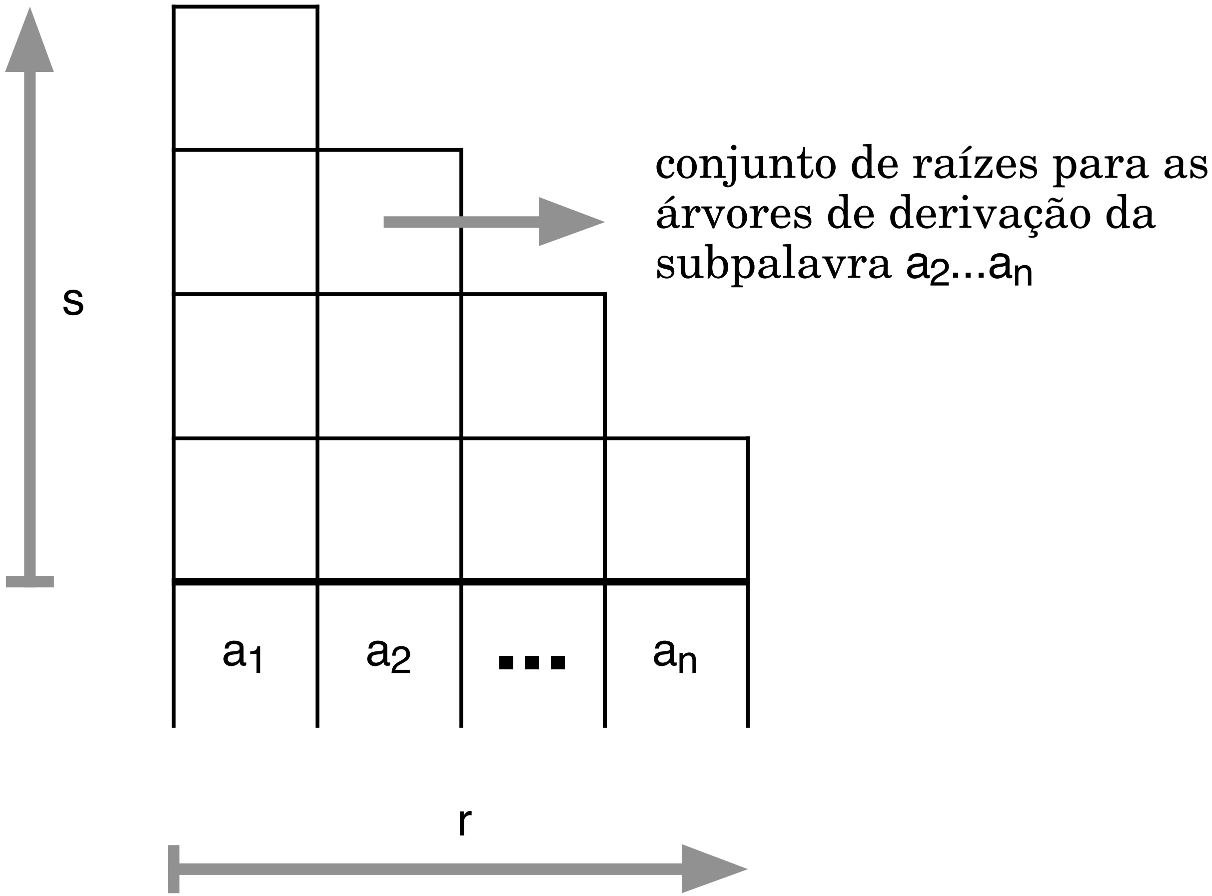
- desenvolvido independentemente por J. Cocke, D. H. Younger e T. Kasami, em 1965

- a partir de uma GLC na Forma Normal de Chomsky
  - gera *bottom-up* todas as árvores de derivação da entrada w
  - tempo de processamento proporcional a $|w|^3$

- idéia básica
  - tabela triangular de derivação
  - célula: raízes que podem gerar a correspondente sub-árvore

Def: Algoritmo de Cocke-Younger-Kasami ou CYK

- $G = (V, T, P, S)$ GLC na Forma Normal de Chomsky
- $w = a_1 a_2 \ldots a_n$ entrada V_{rs} células da tabela



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

para r variando de 1 até n
faça $V_{r1} = \{ A \mid A \rightarrow a_r \in P \}$

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

para s variando de 2 até n
faça **para** r variando de 1 até n - s + 1
    faça $V_{rs} = \varnothing$
        para k variando de 1 até s - 1
        faça $V_{rs} = V_{rs} \cup \{ A \mid A \rightarrow BC \in P,\ B \in V_{rk}$
            e $C \in V_{(r+k)(s-k)} \}$

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

Etapa 3: condição de aceitação da entrada.

- símbolo inicial pertence à V_{1n} (raiz de toda palavra)
  - aceita

Exp: 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

6 – Propriedades e Reconhecimento das LLC

7.1 Propriedades das LLC
- 7.1.1 Investigação se é LLC
- 7.1.2 Operações Fechadas sobre as LLC
- 7.1.3 Investigação se é Vazia, Finita ou Infinita

7.2 Algoritmos de Reconhecimento
- 7.2.1 Autômato com Pilha como Reconhecedor
- 7.2.2 Algoritmo de Cocke-Younger-Kasami
- 7.2.3 Algoritmo de Early

7.1.3 Algoritmo de Early

◆ Algoritmo de Early

- desenvolvido em 1968

- possivelmente o mais rápido algoritmo para LLC

- tempo de processamento proporcional a
  - em geral: $|w|^3$
  - gramáticas não-ambíguas: $|w|^2$
  - muitas gramáticas de interesse prático: $|w|$

◆ Algoritmo *top-down*

- a partir de uma GLC *sem* produções vazias
- parte do símbolo inicial
- executa sempre a derivação mais à esquerda
- cada ciclo gera um terminal
  - comparado com o símbolo da entrada
  - *sucesso* -> construção do conjunto de produções que, potencialmente, pode gerar o próximo símbolo

Def: Algoritmo de Early

- $G = (V, T, P, S)$ GLC sem produções vazias
- $w = a_1 a_2 \ldots a_n$ palavra a ser verificada
- marcador "•"
  - antecedendo a posição, em cada produção, que será analisada
  - na tentativa de gerar o próximo símbolo terminal
- sufixo "/u" adicionado a cada produção
  - indica o u-ésimo ciclo em que passou a ser considerada

Etapa 1: construção de D_0: primeiro conjunto de produções

- produções que partem de S (1)
- produções que podem ser aplicadas (2)
  - em sucessivas derivações mais à esquerda (a partir de S)

```
D0 = ∅
para   toda S → α ∈ P                                            (1)
faça   D0 = D0 ∪ { S → •α/0 }
repita para   toda A → •Bβ/0 ∈ D0                               (2)
    faça   para toda B → φ ∈ P
        faça   D0 = D0 ∪ { B → •φ/0 }
até que o cardinal de D0 não aumente
```

Etapa 2: construção dos demais conjuntos de produção

- $n = |w|$ conjuntos de produção a partir de D_0
- ao gerar a_r, constrói D_r: produções que podem gerar a_{r+1}

```
para   r variando de 1 até n                                    (1)
faça   Dr = ∅;
    para   toda A → α•arβ/s ∈ Dr-1                               (2)
    faça   Dr = Dr ∪ { A → αar•β/s };
    repita
        para   toda A → α•Bβ/s ∈ Dr                              (3)
        faça   para toda B → φ ∈ P
            faça Dr = Dr ∪ { B → •φ/r }
        para   toda A → α•/s de Dr                               (4)
        faça   para toda B → β•Aφ/k ∈ Ds
            faça   Dr = Dr ∪ { B → βA•φ/k }
    até que o cardinal de Dr não aumente
```

(1) cada ciclo gera um conjunto de produções D_r

(2) gera o símbolo a_r

(3) produções que podem derivar o próximo símbolo

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

- ∗ inclui em D_r todas as produções de D_s que referenciaram $\bullet A$;

Etapa 3: condição de aceitação da entrada.

- uma produção da forma $S \rightarrow \alpha \bullet /0$ pertence a D_n
  - w foi aceita

- $S \rightarrow \alpha \bullet /0$ é uma produção que
  - parte do símbolo inicial S
  - foi incluída em D_0 ("$/0$")
  - todo o lado direito da produção foi analisado com sucesso
  - ("$\bullet$" está no final de α)

◆ Otimização do Algoritmo de Early

- ciclos **`repita-até`**
  - ⁎ podem ser restritos exclusivamente às produções recentemente incluídas em D_r ou em D_0 ainda não-analisadas.

Exp: Algoritmo de Early: Expressão Simples

"Expressão simples" da linguagem Pascal

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

Reconhecimento da palavra $x*x$

D_0:

- $E \to \bullet T/0$ — produções que partem
- $E \to \bullet E+T/0$ — do símbolo inicial

- $T \to \bullet F/0$ — produções que podem ser aplicadas
- $T \to \bullet T*F/0$ — em derivação mais à esquerda
- $F \to \bullet [E]/0$ — a partir do símbolo inicial
- $F \to \bullet x/0$

D_1: reconhecimento de x em <u>x</u>∗x

- $F \rightarrow x\bullet/0$ — x foi reduzido a F

- $T \rightarrow F\bullet/0$
- $T \rightarrow T\bullet * F/0$
- $E \rightarrow T\bullet/0$
- $E \rightarrow E\bullet + T/0$

inclui todas as produções de D_0 que referenciaram •F direta ou indiretamente movendo o marcador "•" um símbolo para a direita

D_2: reconhecimento de ∗ em x<u>∗</u>x

- $T \rightarrow T * \bullet F/0$ — gerou ∗; o próximo será gerado por F

- $F \rightarrow \bullet[E]/2$
- $F \rightarrow \bullet x/2$

inclui todas as produções de P que podem gerar o próximo terminal a partir de F•

D_3: reconhecimento de x em $x * \underline{x}$

- $F \to x \bullet /2$ x foi reduzido à F

- $T \to T * F \bullet /0$ incluído de D_2 (pois $F \to x \bullet /2$); entrada reduzida à T
- $E \to T \bullet /0$ incluído de D_0 (pois $T \to T * F \bullet /0$); entrada reduzida à E
- $T \to T \bullet * F/0$ incluído de D_0 (pois $T \to T * F \bullet /0$)
- $E \to E \bullet + T/0$ incluído de D_0 (pois $E \to T \bullet /0$)

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

- entrada aceita

Linguagens Formais e Autômatos

P. Blauth Menezes

Linguagens Formais e Autômatos

P. Blauth Menezes

blauth@inf.ufrgs.br

Departamento de Informática Teórica

Instituto de Informática / UFRGS