

Gramáticas livre-de- contexto

Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática – CEEI

Departamento de Sistemas e Computação – DSC

Professor: Andrey Brito

Período: 2023.2

Forma normal de Chomsky (FNC)

- Uma palavra de comprimento n precisa de **$2*n - 1$ passos de derivação**
- Toda GLC possui uma GLC equivalente na forma normal de Chomsky

Transformação de uma gramática para FNC

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \lambda$

$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$

$S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$

$A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$

$A_1 \rightarrow SA$

$U \rightarrow a$

$B \rightarrow b$

Ambiguidade

Árvore de derivação, derivação mais à esquerda e ambiguidade

- O que a gramática abaixo gera?

$$S \rightarrow S + S \mid S * S \mid a$$

Árvore de derivação, derivação mais à esquerda e ambiguidade

- O que a gramática abaixo gera?

$$S \rightarrow S + S \mid S * S \mid a$$

- Expressões aritméticas (obviamente simplificadas)
- É ambígua?
- Isso é um problema?

Árvore de derivação, derivação mais à esquerda e ambiguidade

- O que a gramática abaixo gera?

$$S \rightarrow S + S \mid S * S \mid a$$

- Expressões aritméticas (obviamente simplificadas)
- É ambígua? **Sim**
- Isso é um problema? **Sim**
- O compilador não somente quer verificar se o programa é válido, mas também quer formar um sentido, uma interpretação para o mesmo
- Neste caso a precedência das operações poderia ser alterada
 - De quantas formas eu poderia gerar $a+a*a$?

Ambiguidade

- Uma palavra pode ser derivada de duas formas diferentes, mesmo usando derivações mais à esquerda
- Regras diferentes foram aplicadas para substituir uma variável
- Note que isso é diferente de aplicar as mesmas sequências de regras para uma variável, mas mudar a ordem que variáveis são substituídas

Ambiguidade (2)

- A gramática abaixo gera expressões de uma forma não ambígua
- Não é possível gerar $a+a*a$ de duas formas diferentes
 - (Onde “a” é um dígito, número ou identificador da linguagem de programação)

$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERMO} \rangle \mid \langle \text{TERMO} \rangle$
 $\langle \text{TERMO} \rangle \rightarrow \langle \text{TERMO} \rangle \times \langle \text{FATOR} \rangle \mid \langle \text{FATOR} \rangle$
 $\langle \text{FATOR} \rangle \rightarrow a$

Ambiguidade (3)

- Outro exemplo:
 - $S \rightarrow SS \mid (S) \mid ()$
 - É ambígua! Como eu poderia gerar “() $()()$ ”?
- Mas essa não:
 - $B \rightarrow (RB \mid \lambda$
 - $R \rightarrow) \mid (RR$

Ambiguidade (4)

- Outro exemplo

<CMD> → if <COND> then <CMD> | if <COND> then <CMD> else <CMD>

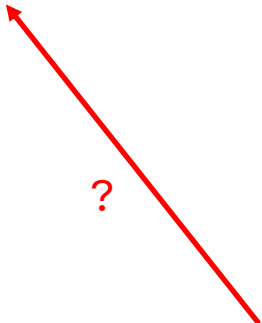
Ambiguidade (4)

$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERMO} \rangle \mid \langle \text{TERMO} \rangle$
 $\langle \text{TERMO} \rangle \rightarrow \langle \text{TERMO} \rangle \times \langle \text{FATOR} \rangle \mid$
 $\langle \text{FATOR} \rangle$
 $\langle \text{FATOR} \rangle \rightarrow a$

- Outro exemplo

$\langle \text{CMD} \rangle \rightarrow \text{if } \langle \text{COND} \rangle \text{ then } \langle \text{CMD} \rangle \mid \text{if } \langle \text{COND} \rangle \text{ then } \langle \text{CMD} \rangle \text{ else } \langle \text{CMD} \rangle$

$S \rightarrow S + S \mid S * S \mid a$



?

- “Difícil” de detectar e de consertar, além de que algumas linguagens são sempre ambíguas
 - Pode ter conserto ou talvez linguagem **possa ser modificada sem grande prejuízo**
 - Algumas ferramentas para geração de compiladores deixam especificar precedência de operadores como forma de tratar ambiguidades, pode ser mais simples

Equivalência entre APs e GLCs

Equivalência do autômato de pilha

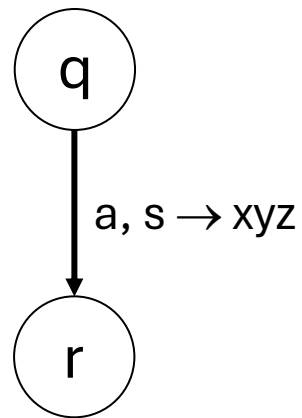
- Parte 1: Se uma linguagem é livre-de-contexto então ela é reconhecida por um autômato de pilha P
- Ideia
 - P é construído a partir da gramática que define a linguagem
 - P tem que aceitar a entrada w se existe alguma série de substituições usando regras da gramática que leve do símbolo inicial à w
- Intuição: a pilha é o que se espera ver na entrada

Autômato de pilha (extensão de δ)

- Quero permitir uma função de transição que faça: $\delta'(q,a,s) = (r, xyz)$

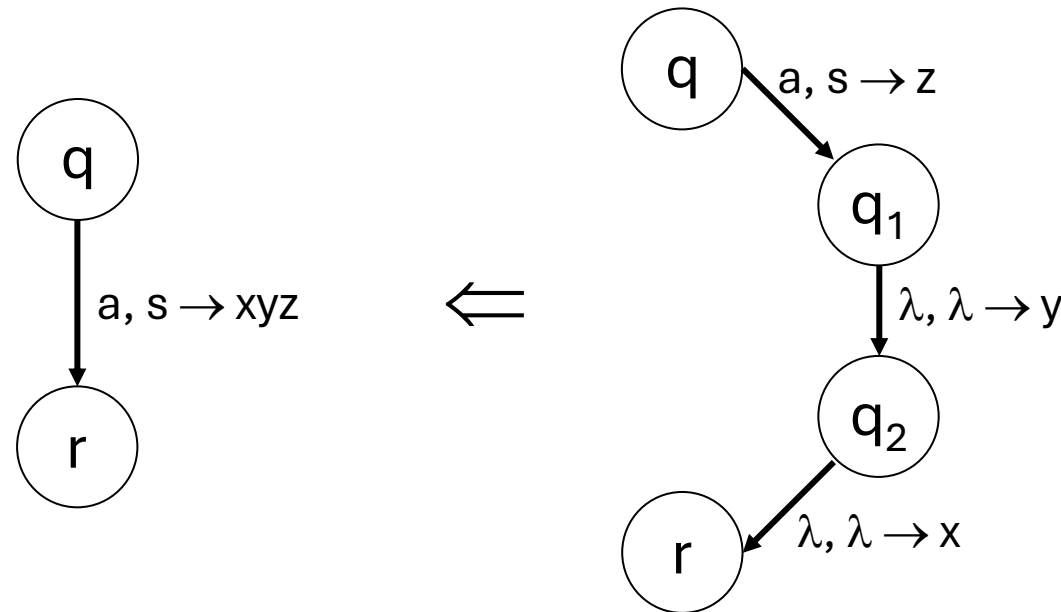
Autômato de pilha (extensão de δ)

- Quero permitir uma função de transição que faça: $\delta'(q, a, s) = (r, xyz)$



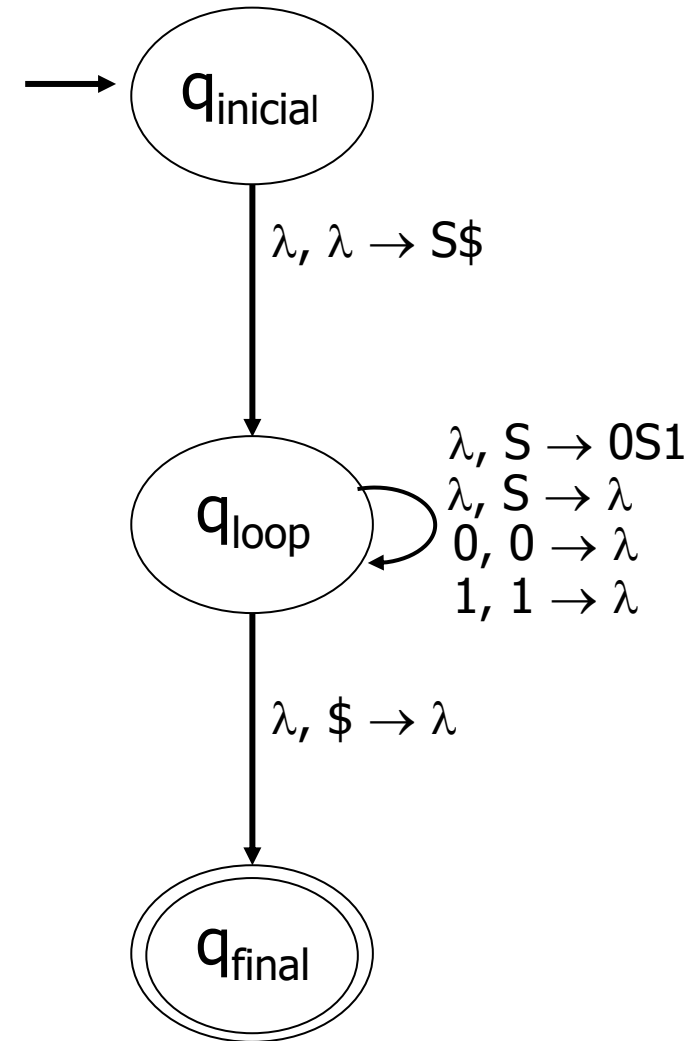
Autômato de pilha (extensão de δ)

- Quero permitir uma função de transição que faça: $\delta'(q, a, s) = (r, xyz)$

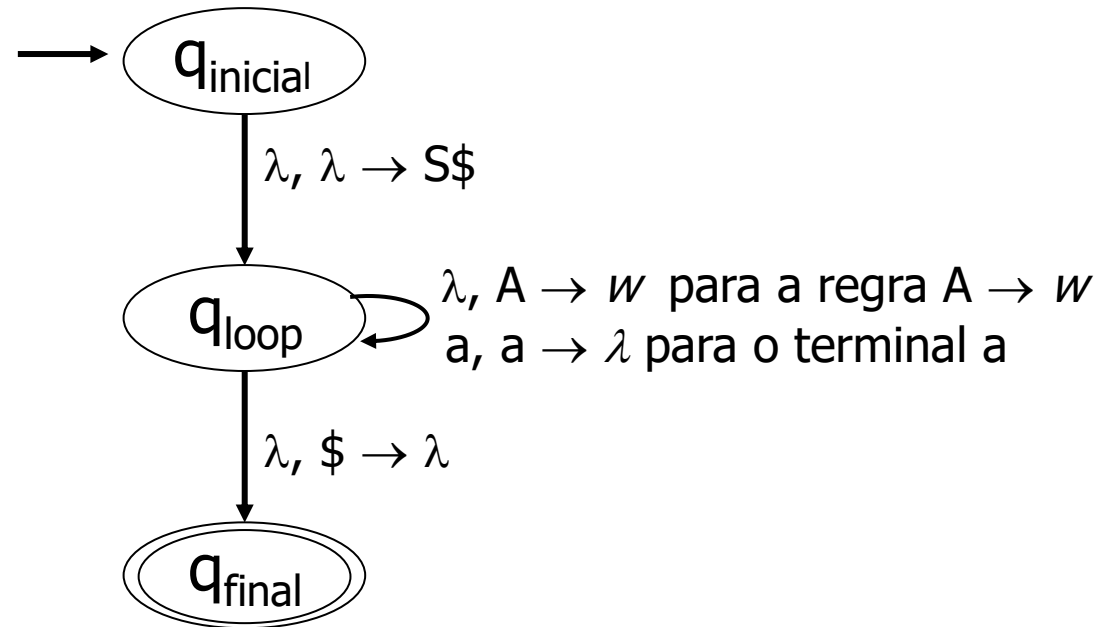


Exemplo

- GLC G_1 :
 $S \rightarrow 0S1 \mid \lambda$
- $L(G_1)$: $0^n 1^n$



Equivalência do autômato de pilha



$$\delta(q_{\text{inicial}}, \lambda, \lambda) = \{ (q_{\text{loop}}, S\$) \}$$

$$\delta(q_{\text{loop}}, \lambda, A) = \{ (q_{\text{loop}}, w) \mid A \rightarrow w \in R \}$$

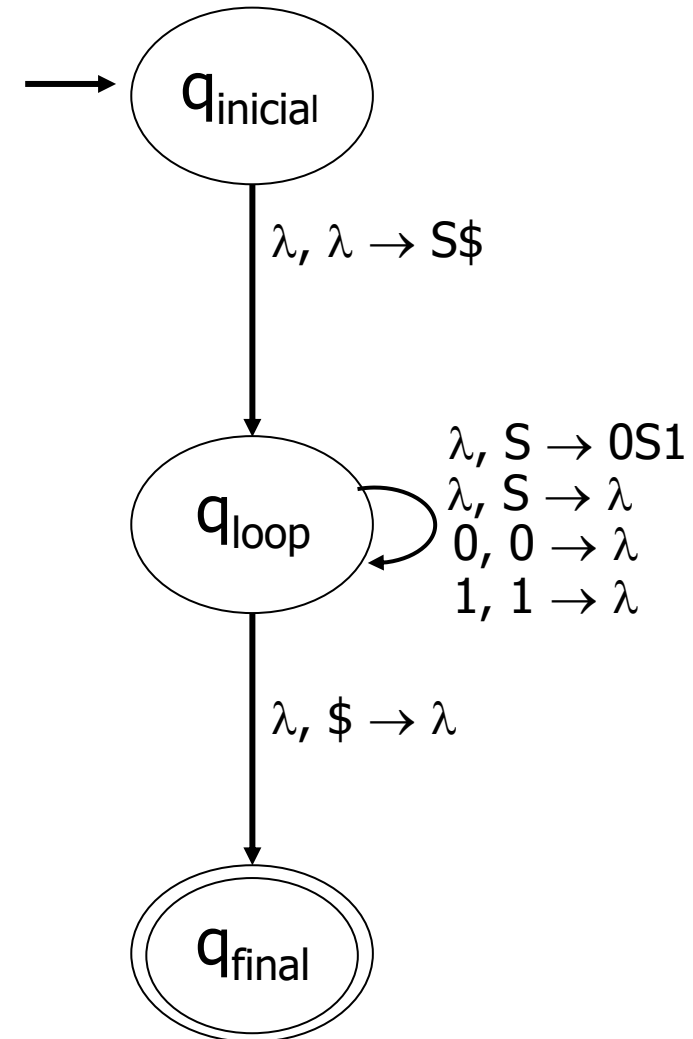
$$\delta(q_{\text{loop}}, a, a) = \{ (q_{\text{loop}}, \lambda) \}$$

$$\delta(q_{\text{loop}}, \lambda, \$) = \{ (q_{\text{aceita}}, \lambda) \}$$

Exemplo

- GLC G_1 :
 $S \rightarrow 0S1 \mid \lambda$
- $L(G_1)$: $0^n 1^n$

$$\begin{aligned}\delta(q_{\text{inicio}}, \lambda, \lambda) &= \{(q_{\text{loop}}, S\$)\} \\ \delta(q_{\text{loop}}, \lambda, S) &= \{(q_{\text{loop}}, 0S1), (q_{\text{loop}}, \lambda)\} \\ \delta(q_{\text{loop}}, 0, 0) &= \{(q_{\text{loop}}, \lambda)\} \\ \delta(q_{\text{loop}}, 1, 1) &= \{(q_{\text{loop}}, \lambda)\} \\ \delta(q_{\text{loop}}, \lambda, \$) &= \{(q_{\text{final}}, \lambda)\}\end{aligned}$$



Equivalência do autômato de pilha

- Colocar a cadeia $S\$$ na pilha, onde S é a variável inicial
- Repetir
 - Se o topo da pilha é uma variável A , selecionar (não-deterministicamente) uma das regras para A , $(A \rightarrow w)$, e substituir A na pilha pelo lado direito da regra (w)
 - Se o topo da pilha é um terminal a , ler a entrada e compará-la com a . Se “casa”, desempilhar a e repetir. Se não, rejeitar esse ramo do não-determinismo
 - Se o topo da pilha é $\$$ entrar no estado de aceitação. A palavra é aceita se a entrada acabou

Equivalência do autômato de pilha - Ideia

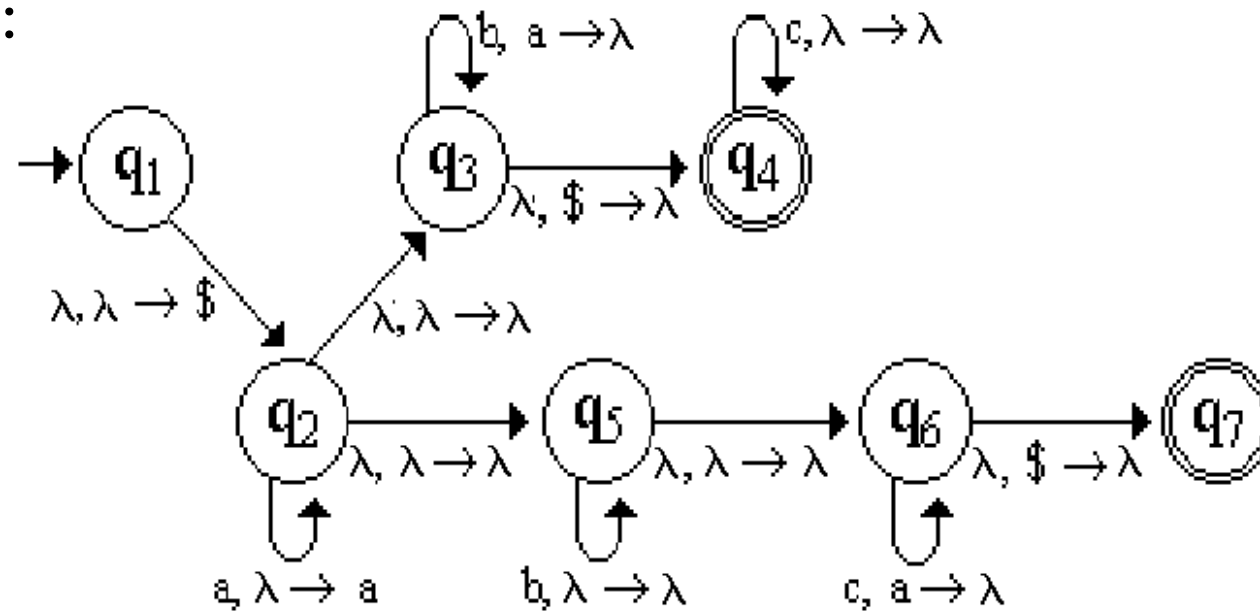
- Se a palavra passada ao AP pertence à linguagem dele, os símbolos devem aparecer da “forma que ele espera”
- O que é a “forma que ele espera”?
 - Qualquer coisa que pode ser gerada pela gramática
 - E se a gramática gerasse apenas uma palavra?
 - $S \rightarrow 00$
 - As vezes mais de um “próximo símbolo” é possível, então como tratar?
 - $S \rightarrow 0A1$
 - $A \rightarrow 0 \mid 1$

Equivalência do autômato de pilha

- Interpretações
 - Quando mais de uma substituição é possível, você deve lembrar do não determinismo: o autômato testa em paralelo todas as possibilidades
 - O que significa dizer que duas possíveis execuções aceitam?
 - Se dois ramos sobreviverem é “sinal” de ambiguidade
 - Por que???
 - Talvez desse para fazer um autômato determinístico, talvez não
 - Existem linguagens que só podem ser geradas por gramáticas ambíguas
 - $L = a^i b^j c^k$ onde $i=j$ ou $i=k$

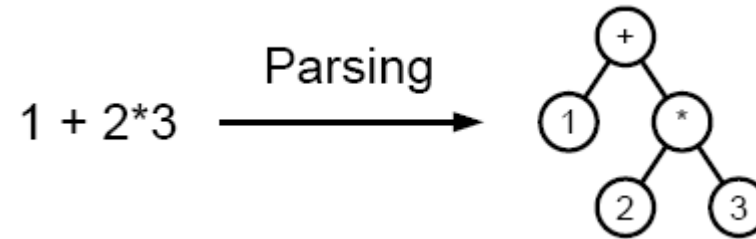
Relembrando...

P_2 :



$$L(P_2) = ?$$

Exemplo da relação TC – Compiladores



Exemplo da **análise sintática** de uma [expressão matemática](#). O resultado é uma [árvore](#) da expressão

https://pt.wikipedia.org/wiki/An%C3%A1lise_sint%C3%A1tica_%28computa%C3%A7%C3%A3o%29

Exemplo: Analisador sintático (1/2)

https://pt.wikipedia.org/wiki/Analisador_sint%C3%A1tico_LL

Caso geral [[editar](#) | [editar código-fonte](#)]

O analisador sintático trabalha em cadeias de texto de uma determinada [gramática formal](#), e consiste de:

- um [buffer](#) de entrada;
- uma [pilha](#) na qual são armazenados os símbolos da gramática ainda não analisados;
- uma [tabela análise](#) que indica se qual [regra gramatical](#) a ser aplicada dados os símbolos no topo da pilha e o próximo token de entrada.

Quando o analisador é iniciado, a pilha já contém dois símbolos:

```
[ S, $ ]
```

no qual **\$** é um terminador especial para indicar o fim da pilha e o fim da entrada de dados, e **S** é o símbolo de entrada da gramática. O analisador sintático irá tentar reescrever o conteúdo da pilha para o que ele interpreta da entrada de texto. Entretanto, ele somente mantém na pilha o que ainda deve ser reescrito.

Exemplo: Analisador sintático (2/2)

Exemplo [\[editar | editar código-fonte \]](#)

A gramática abaixo será usada para o exemplo a seguir. Ela trata expressões matemáticas, no qual são aceitas somas entre uns:

$$(1) S \rightarrow F$$

$$(2) S \rightarrow (S + F)$$

$$(3) F \rightarrow 1$$

deve-se analisar sintaticamente a seguinte entrada:

(1 + 1)

Tabela de análise [\[editar | editar código-fonte \]](#)

	()	1	+	\$
S	2	-	1	-	-
F	-	-	3	-	-

Note que a tabela é simplesmente uma indicação de que transição foi usada.