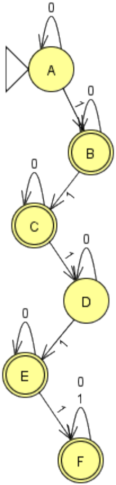


Teoria de Linguagens e Compiladores
Linguagens Livres de Contexto

Luiz Eduardo da Silva

Universidade Federal de Alfenas

17 de Fevereiro de 2021

Agenda

Agenda

Introdução

- Já vimos que linguagens como $\{0^n1^n | n \geq 0\}$ não podem ser descritas usando **linguagens regulares**.
- Essa linguagens podem ser descritas por **gramáticas livres de contexto**, que é um método mais poderoso para descrever linguagens.
- Essa método é usado para descrever e analisar as construções sintáticas encontradas na maioria das linguagens de programação.
- As **gramáticas livres de contexto** descrevem **linguagens livres de contexto**
- Os **autômatos de pilha** são uma classe de máquinas que reconhecem as linguagens livres de contexto.

Agenda

Exemplo de GLC

$$\begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \\ B &\rightarrow \# \end{aligned}$$

- Uma **gramática** consiste de uma coleção de **regras** (também chamadas de **produções**).
- O símbolo à esquerda de cada regra é a **variável**.
- A sequência no lado direito das regras é formada por variáveis e símbolos **terminais**.
- Uma variável é designada **variável inicial**(ou de **partida**).
- Nesses exemplo: As variáveis são A e B. Os terminais são 0, 1 e $\#$.

Exemplo de GLC

$$
\begin{array}{rcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

- Usa-se a gramática G para **gerar** todas as cadeias da linguagem $L(G)$, da seguinte maneira:
  1. Escreve-se a variável inicial. Normalmente a variável à esquerda da primeira regra.
  2. Encontre uma variável e uma regra que pode substituir essa variável, trocando pela forma sentencial no lado direito dessa regra.
  3. Repita o passo 2, até que não reste nenhuma variável a forma sentencial.

Exemplo de GLC

$$\begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \\ B &\rightarrow \# \end{aligned}$$

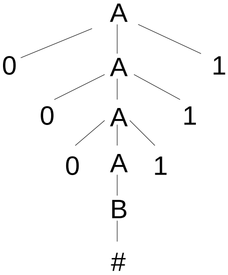
- A sequência de substituições para gerar uma cadeia é chamada de **derivação** (representada pelo símbolo $\Rightarrow$). Uma derivação da cadeia 000#111 é:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

- Outra forma de representar essa sequência de substituições é através da construção de uma **árvore de derivação**.

Exemplo de GLC

$$
\begin{aligned}
A &\rightarrow 0A1\\
A &\rightarrow B\\
B &\rightarrow \#
\end{aligned}
$$



Definição

Definição

Uma gramática livre de contexto (GLC) é uma 4-upla $G = (V, \Sigma, R, P)$*, onde:*

- V *é o conjunto finito das* variáveis.
- Σ *é o conjunto finito, disjunto de* V*, denominado* terminais
- R *é o conjunto de regras, cada regra tem a forma* $X \rightarrow w$*, em que* $X \in V$ e $w \in (V \cup \Sigma)^*$.
- $P \in V$ *é o símbolo* inicial *ou de* partida.

A **linguagem da gramática**, $L(G) = \{w \in \Sigma^* | P \overset{*}{\Rightarrow} w\}$

Exemplo

- As GLC´s são usadas para especificar, através de definições indutivas, todas as construções sintáticas válidas para as linguagens de programação.
- Usamos GLC para especificar a sintaxe das linguagens que são reconhecidas pelo analisador sintático de compilador.

Exemplo:

$$G = (\{C, E, I, O\}, \{a, b, +, *, (,), \underline{:=}, \underline{;}\}, R, C\}$$

onde:

$$R = \left\{ \begin{array}{ccl} C & \rightarrow & I := E | C ; C \\ E & \rightarrow & I | EOE | (E) \\ O & \rightarrow & + | * \\ I & \rightarrow & a | b \end{array} \right\}$$

Outro exemplo

Considere $G_3 = (\{S\}, \{a, b\}, R, S)$, onde o conjunto das regras R é:

$$\begin{array}{lcl} S & \rightarrow & aSb \\ S & \rightarrow & SS \\ S & \rightarrow & \varepsilon \end{array}$$

- Essa linguagem gera cadeias como $abab$, $aaabbb$ e $aababb$.
- Se pensar a como parêntese à esquerda "(" e b como ")".
- $L(G_3)$ é a linguagem de parênteses apropriadamente aninhados.

Projeto

- Assim como AFD, o projeto de GLCs requer criatividade.
- Algumas dicas:
  1. LLCs são normalmente união de LLCs mais simples. Exemplo: $L = \{0^n1^n | n \geq 0\} \cup \{1^n0^n | n \geq 0\}$ constrói-se $S_1 \rightarrow 0S_11|\varepsilon$ e $S_2 \rightarrow 1S_20|\varepsilon$ e então adiciona-se $S \rightarrow S_1|S_2$.
  2. Construir o GLC para uma linguagem regular é fácil se já tem o AFD:
     - Pegue uma variável R_i para cada estado q_i do AFD.
     - Adicione $R_i \rightarrow aR_j$ na GLC, se tem $\delta(q_i, a) = q_j$ no AFD.
     - Adicione $R_i \rightarrow \varepsilon$ se q_i for estado de aceitação.
     - Faça R_0 a variável de partida, se q_0 é o estado inicial do AFD.
  3. Linguagens da forma $L = \{0^n1^n | n \geq 0\}$ pode ser construída com regras da forma $R \rightarrow uRv$, onde os u's correspondem aos v's.
  4. Linguagens mais complexas podem ocorrer estruturas que aparecem recursivamente.

Ambiguidade

Definição

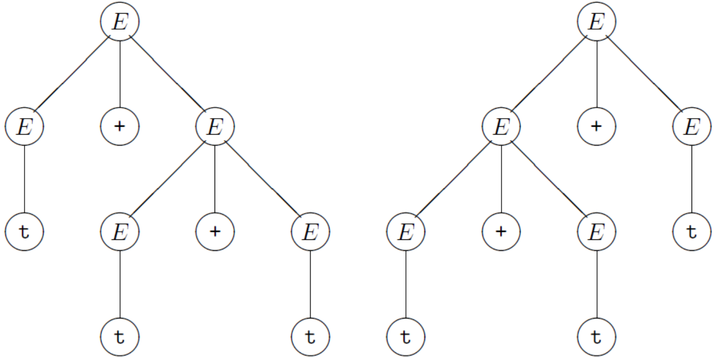
Uma gramática livre de contexto (GLC) é denominada ambígua quando existe mais de uma árvore de derivação (AD) para alguma sentença que a gera.

Ambiguidade

Para a gramática:

$E \rightarrow E + E | E * E | (E) | t$

Temos duas árvores de derivação distintas para $t + t + t$:



Quando se trabalha com GLCs é conveniente tê-as numa forma simplificada.

Definição

Uma GLC está na forma normal de chomsky *se toda regra é da forma:*

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

onde

- a é qualquer terminal
- A, B e C, são variáveis.
- B e C não podem ser variáveis iniciais.
- Permitimos a regra $S \rightarrow \varepsilon$, se S é a variável inicial.

Teorema

Teorema

Qualquer GLC é gerada por uma GLC na forma normal de chomsky

Teorema

- Prova:
  1. Adiciona-se uma nova variável inicial $S_0 \rightarrow S$ (para S inicial). Garante que S_0 não ocorre no lado direito das regras.
  2. Removemos as regras ε. Se $A \rightarrow \varepsilon$ então:
     - Se $R \rightarrow uAv$, adicionamos $R \rightarrow uv$
     - Se $R \rightarrow uAvAw$, adicionamos $R \rightarrow uvAw$, $R \rightarrow uAvw$ e $R \rightarrow uvw$.
     - Se $R \rightarrow A$, adicionamos $R \rightarrow \varepsilon$ e fazemos o passo 2 para a variável R, até eliminarmos todas as regras ε.
  3. Removemos tods as regras unitárias $A \rightarrow B$. Se $B \rightarrow u$ então $A \rightarrow u$
  4. Convertemos todas as regras para a forma apropriada:
     - Substituímos cada regra $A \rightarrow u_1 u_2 ... u_k$, onde $k > 3$ e cada u_i é um terminal ou variável, por $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2$, ..., $A_{k-2} \rightarrow u_{k-1} u_k$. Os A_is são novas variáveis.
     - Se $k = 2$, substituímos os terminais u_i pela nova variável U_i e acrescentamos a regra $U_i \rightarrow u_i$

Transformação Forma Normal de Chomsky

- Adiciona-se S_0:

$$\begin{aligned} S &\rightarrow ASA|aB \\ A &\rightarrow B|S \\ B &\rightarrow b|\varepsilon \end{aligned}$$

$$\begin{aligned} \mathrm{S}_0 &\rightarrow \mathrm{S} \\ S &\rightarrow ASA|aB \\ A &\rightarrow B|S \\ B &\rightarrow b|\varepsilon \end{aligned}$$

- Remova $B \rightarrow \varepsilon$ e $A \rightarrow \varepsilon$:

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA|aB|\mathrm{a} \\ A &\rightarrow B|S|\varepsilon \\ B &\rightarrow b|\cancel{\varepsilon} \end{aligned}$$

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA|aB|a|\boldsymbol{SA}|\boldsymbol{AS}|\boldsymbol{S} \\ A &\rightarrow B|S|\cancel{\varepsilon} \\ B &\rightarrow b \end{aligned}$$

Transformação Forma Normal de Chomsky

- Remova regras unitárias $S \to S$ e $S_0 \to S$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA|aB|a|SA|AS|\cancel{S} \\
A &\to B|S \\
B &\to b
\end{aligned}
\qquad
\begin{aligned}
S_0 &\to \cancel{S}|\boldsymbol{ASA}|\boldsymbol{aB}|\boldsymbol{a}|\boldsymbol{SA}|\boldsymbol{AS} \\
S &\to ASA|aB|a|SA|AS \\
A &\to B|S \\
B &\to b
\end{aligned}
$$

- Remova as regras unitárias $A \to B$ e $A \to S$:

$$
\begin{aligned}
S_0 &\to ASA|aB|a|SA|AS \\
S &\to ASA|aB|a|SA|AS \\
A &\to \cancel{B}|S|\boldsymbol{b} \\
B &\to b
\end{aligned}
\qquad
\begin{aligned}
S_0 &\to ASA|aB|a|SA|AS \\
S &\to ASA|aB|a|SA|AS \\
A &\to \cancel{S}|b|\boldsymbol{ASA}|\boldsymbol{aB}|\boldsymbol{a}|\boldsymbol{SA}|\boldsymbol{AS} \\
B &\to b
\end{aligned}
$$

Transformação Forma Normal de Chomsky

$$
\begin{array}{rcl}
S_0 & \rightarrow & ASA|aB|a|SA|AS \\
S & \rightarrow & ASA|aB|a|SA|AS \\
A & \rightarrow & b|ASA|aB|a|SA|AS \\
B & \rightarrow & b
\end{array}
$$

- Acrescente variáveis e regras adicionais:

$$
\begin{array}{rcl}
S_0 & \rightarrow & AA_1|UB|a|SA|AS \\
S & \rightarrow & AA_1|UB|a|SA|AS \\
A & \rightarrow & b|AA_1|UB|a|SA|AS \\
A_1 & \rightarrow & SA \\
U & \rightarrow & a \\
B & \rightarrow & b
\end{array}
$$

Agenda

Definição

Definição

Seja uma GLC $G = (V, \Sigma, R, P)$. Uma variável $X \in V$ é dita ser útil se, e somente se, existem $u, v \in (V \cup \Sigma)^*$ e $w \in \Sigma^*$ tais que

$$P \stackrel{*}{\Rightarrow} uXv \stackrel{*}{\Rightarrow} w$$

Exemplo

Exemplo

Seja a Gramática:

- $P \rightarrow AB|a$
- $B \rightarrow b$
- $C \rightarrow c$

Então:

- C *é inútil porque não existem* u *e* v *tais que* $P \overset{*}{\Rightarrow} uCv$;
- A *é inútil porque não existe* $w \in \Sigma^*$ *tal que* $A \overset{*}{\Rightarrow} w$;
- B *é inútil porque* $P \overset{*}{\Rightarrow} uBv$, *apenas para* $u = A$ *e* $v = \lambda$, *e não existe* $w \in \Sigma^*$ *tal que* $AB \overset{*}{\Rightarrow} w$.

Outras definições

Definição

Uma variável X é dita ser anulável em uma GLC se e somente se, $X \stackrel{*}{\Rightarrow} \lambda$.

Definição

Para qualquer GLC, existe um GLC equivalente cuja única regra λ, se houver é $P \rightarrow \lambda$, sendo P é o simbolo de Partida.

Outras definições

Definição

Para qualquer GLC, existe um GLC equivalente sem regras unitárias. Uma GLC equivalente a $G = (V, \Sigma, R, P)$ *seria* $G' = (V, \Sigma, R', P)$, *em que:* $R' = \{X \rightarrow w |$ *existe* $Y \in enc(X)$ *tal que* $Y \rightarrow w \in R$ *e* $w \notin V\}$

Outras definições

Exemplo

A gramática:

$E \to E + T|T$

$T \to T * F|F$

$F \to (E)|t$

Onde o conjunto das variáveis encadeadas são:

- $enc(E) = \{E, T, F\}$
- $enc(T) = \{T, F\}$
- $enc(F) = \{F\}$

Fica (sem regras unitárias):

$E \to E + T|T * F|(E)|t$

$T \to T * F|(E)|t$

$F \to (E)|t$

Outras definições

Definição

Para qualquer GLC, existe um GLC $G = (V, \Sigma, R, P)$, *existe uma GLC equivalente, cujas regras são da forma:*

- $P \rightarrow \lambda$ *se* $\lambda \in L(G)$
- $X \rightarrow a$ *para* $a \in \Sigma$
- $X \rightarrow w$ *para* $|w| \geq 2$

Outras definições

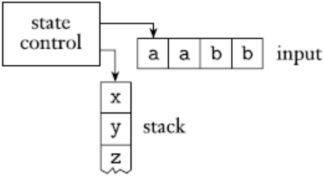
Definição

Para qualquer GLC, existe uma GLC equivalente sem regras recursivas à direita.

- Todas as regras X de uma GLC, da forma:
  $X \rightarrow Xy_1|Xy_2|...|Xy_n|w_1|w_2|...|w_k$
- Podem ser substituídas por recursões à direita:
  $X \rightarrow w_1Z|w_2Z|...|w_kZ$
  $Z \rightarrow y_1Z|y_2Z|...|y_nZ|\lambda$
- **Exemplo:** A gramática: $E \rightarrow E + E|E * E|(E)|t$, sem recursões à esquerda fica:
  $E \rightarrow (E)Z|tZ$
  $Z \rightarrow +EZ| * EZ|\lambda$

Agenda

AP



- O **autômato com pilha** é um autômato finito não-determinístico que tem um componente extra, a **pilha**.
- Tem poder equivalente a GLCs.
- A pilha é "infinita" e símbolos podem ser empilhados e/ou desempilhados nas transições.
- Autômato de Pilha Determinístico **NÃO** é equivalente a Autômato de Pilha Não-Determinístico.

Definição

Definição

Um autômato de pilha *é uma 6-upla* $A = (Q, \Sigma, \Gamma, \delta, q_0, F)$, *onde* $Q, \Sigma, \Gamma, \delta$ *e* F *são todos conjuntos finitos, e*

1. Q *é o conjunto de estados,*
2. Σ *é o alfabeto de entrada,*
3. Γ *é o alfabeto da pilha,*
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ *é a função de transição*
5. $q_0 \in Q$ *é o estado inicial*
6. $F \subseteq Q$ *é o conjunto de estados de aceitação.*

Exemplo de AP

Seja $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ e $L(M_1) = \{0^n 1^n | n \geq 0\}$

- $Q = \{q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, \$\}$,
- $F = \{q_1, q_4\}$ e
- δ é dada por:

Entrada:	0			1			ε		
Pilha:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Diagrama do AP

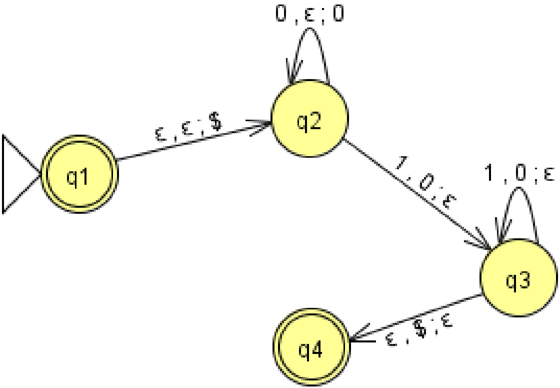
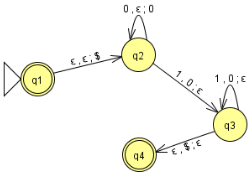


Diagrama do AP

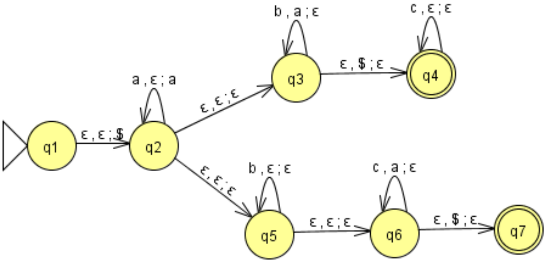


- O diagrama do **AP** é semelhante ao diagrama de estados usados para descrever **AFD** e **AFN**.
- Nas transições escrevemos "a,b;c" (ou "a,$b \rightarrow c$", conforme livro do Sipser), que significa que a máquina está lendo **a** da entrada e substituindo **b** por **c** no topo da pilha.
- **a**, **b** ou **c** podem ser ε.
  - $a = \varepsilon$, faz transição sem ler da entrada.
  - $b = \varepsilon$, faz transição sem desempilhar.
  - $c = \varepsilon$, faz transição sem empilhar.

Outro exemplo de AP

Diagrama de estados para AP M_2 que reconhece

$$L(M_2) = \{a^i b^j c^k | i, j, k \geq 0 \text{ e } i = j \text{ ou } i = k\}$$

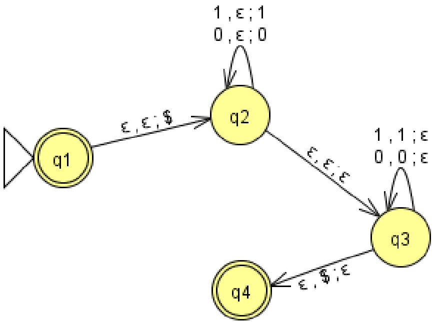


O não determinismo é **essencial** para implementação do **AP** para essa linguagem.

Outro exemplo de AP

Diagrama de estados para AP M_3 que reconhece

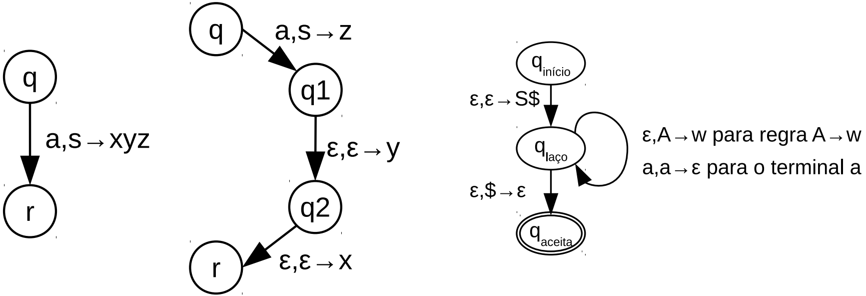
$$L(M_3) = \{ww^R | w \in \{0,1\}^*\}$$



GLCs $\leftrightarrow$ AP

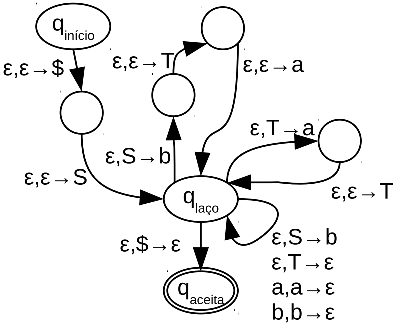
Teorema

Uma linguagem é livre de contexto *se e somente se algum* autômato de pilha *a reconhece.*



GLC → AP - Exemplo

$$
\begin{aligned}
S &\rightarrow aTb|b \\
T &\rightarrow Ta|\varepsilon
\end{aligned}
$$

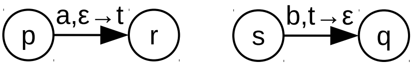


AP $\rightarrow$ GLC

Prova

Seja $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{aceita}\})$ *. Então* $G = (V, \Sigma, R, P)$ *pode ser construído como:*

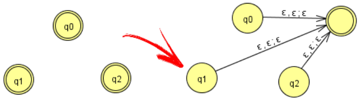
- $V = \{A_{pq} | p, q \in Q\}$,
- $P = A_{q_0, q_{aceita}}$,
- *E as regras R são construídas como:*
  - *Para todo* $p, q, r, s \in Q$, $t \in \Gamma$ *e* $a, b \in \Sigma_\varepsilon$, *se* $\delta(p, a, \varepsilon) \supset (r, t)$ *e* $\delta(s, b, t) \supset (q, \varepsilon)$, *ponha a regra* $A_{pq} \rightarrow aA_{rs}b$ *em G.*
  - *Para todo* $p, q, r \in Q$, *ponha a regra* $A_{pq} \rightarrow A_{pr}A_{rq}$ *em G.*
  - *Para todo* $p \in Q$, *ponha a regra* $A_{pp} \rightarrow \varepsilon$ *em G.*



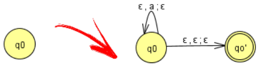
$A_{pq} \rightarrow aA_{rs}b$

Condições - AP Simplificado

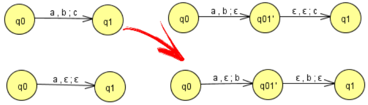
- O AP deve ter somente um estado de aceitação



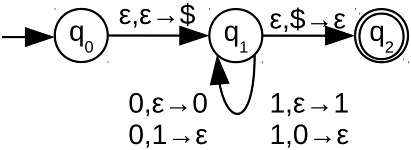
- A pilha deve estar vazia antes de aceitar



- Cada transição deve empilhar ou desempilhar, mas não ambas

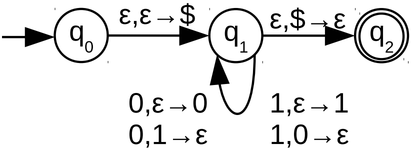


AP → GLC - Exemplo



- Variáveis:

AP → GLC - Exemplo

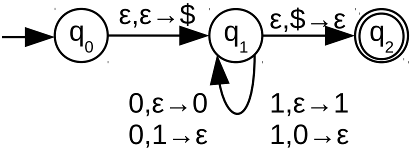


- Variáveis:

$$A_{00}, A_{11}, A_{22}, A_{01}, A_{02}, A_{12}$$

- Variável de Partida:

AP $\rightarrow$ GLC - Exemplo



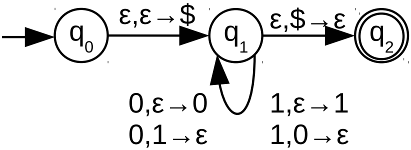
- Variáveis:

$$A_{00}, A_{11}, A_{22}, A_{01}, A_{02}, A_{12}$$

- Variável de Partida:

$$A_{02}$$

AP $\rightarrow$ GLC - Exemplo



- Variáveis:

$$A_{00}, A_{11}, A_{22}, A_{01}, A_{02}, A_{12}$$

- Variável de Partida:

$$A_{02}$$

$$A_{02} \rightarrow A_{01}A_{12}$$
$$A_{01} \rightarrow A_{01}A_{11}$$
$$A_{12} \rightarrow A_{11}A_{12}$$
$$A_{11} \rightarrow A_{11}A_{11}$$
$$A_{11} \rightarrow 0A_{11}1$$
$$A_{11} \rightarrow 1A_{11}0$$
$$A_{02} \rightarrow A_{11}$$
$$A_{00} \rightarrow \varepsilon$$
$$A_{11} \rightarrow \varepsilon$$
$$A_{22} \rightarrow \varepsilon$$

Agenda

Lema do bombeamento

Definição

Se A é uma LLC, então existe p (o comprimento do bombeamento), onde se $s \in A$, então s pode ser dividida em 5 partes $s = uvxyz$, satisfazendo as seguintes condições:

1. *para cada $i \geq 0$, $uv^ixy^iz \in A$,*
2. $|vy| > 0$ *e,*
3. $|vxy| \leq p$.

Exemplo de linguagem não livre de contexto

$L = a^i b^i c^i | i > 0$ não é uma linguagem livre de contexto

A prova por contradição

- Suponha que L é LLC. Então existe p e considere a cadeia $s = a^p b^p c^p \in L$
- Pela escolha de s e o fato que $|vxy| \leq p$, então vxy não pode conter mais de duas letras distintas. Temos 5 possibilidades para vxy:
  - $vxy = a^j$ para $j \leq p$.
  - $vxy = a^j b^k$ para j e k com $j + k \leq p$.
  - $vxy = b^j$ para $j \leq p$.
  - $vxy = b^j c^k$ para j e k com $j + k \leq p$.
  - $vxy = c^j$ para $j \leq p$.
- Para cada caso, é fácil verificar que uv^2xy^2z, por exemplo, não tem a forma $a^i b^i c^i$.

Agenda

Determinismo e Análise Sintática

- GLCs são usadas para modelar sintaxe de linguagens de programação (LP)
- O analisador sintático (AS) do compilador da LP deve decidir se a cadeia (programa) segue as regras de sintaxe (da gramática) ou não.
- Podemos implementar o AS usando Autômatos de Pilha não determinístico, mas essa soluçao não parece muito eficiente.
- Felizmente, existem gramáticas especiais livres de contexto, que se adequam aos problemas sintáticos das LPs e que podem ser implementadas usando autômatos de pilha determinísticos

Árvore Sintática

- A análise sintática é executada pelo *parser* e sua função principal é agrupar os *tokens*, retornados do analisar léxico, em estruturas sintáticas (comando, bloco, expressão, identicador, número, etc).
- Uma estrutura que pode ser empregada é a **árvore sintática**. A árvore representa a aplicação das regras sintáticas da linguagem e definem, de certa forma, um significado para a estrutura do programa compilado.
- O programa está sintaticamente correto se for possível construir uma única árvore sintática, no qual a raiz é o símbolo inicial da gramática da linguagem (símbolo de partida) e as folhas são os tokens retornados do analisado léxico.

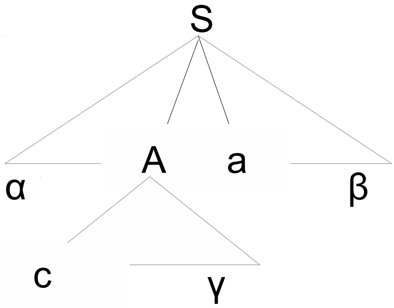
Construção da AS

- Existem duas importantes estratégias para o problema de análise sintática:
  - Análise Sintática Ascendente
  - Análise Sintática Descendente
- Na primeira, a árvore sintática é construída partindo-se da cadeia a ser analisada, "subindo-se" até atingir o símbolo inicial da gramática.
- Na segunda, parte-se do símbolo inicial e vai-se "descendo" até atingir todos os símbolos da cadeia que está sendo analisada.

LL

- Os analisadores descendentes (*top-down*) podem ser construídos com uma classe de gramática chamada LL(1).
  - O primeiro "L"se refere a forma como é lida a cadeia de entrada na análise, nesse caso da esquerda para direita (*Left-to-right*)
  - O segundo "L"se refere a forma como é obtida a sequência de derivação para obtenção da sentença avaliada, nesse caso derivação mais à esquerda (*leftmost*)
  - O número "1"se refere ao número de símbolos a frente deve-se olhar para decidir que regra da gramática deve ser utilizada.
  - Duas funções aplicadas sobre a gramática LL (FIRST e FOLLOW) ajudam a construir uma tabela de análise LL(1) que determina de forma preditiva que regra de substituição usar a cada passo da análise.

Conjuntos FIRST e FOLLOW



- Intuitivamente, todos os símbolos que iniciam derivações de A compõe o conjunto *FIRST* de A (por exemplo, o terminal c da Figura)
- Todo símbolo terminal que segue o símbolo A em qualquer derivação faz parte do conjunto *FOLLOW* de A (por exemplo, o terminal a da Figura)

Conjunto FIRST

Para calcular $FIRST$ de todos os simbolos X de uma gramática, execute as seguintes regras até que nenhum novo símbolo possa ser acrescentado a qualquer conjunto $FIRST$.

1. Se X é um símbolo terminal, então $FIRST(X) = \{X\}$
2. Se X é um não-terminal e $X \rightarrow Y_1 Y_2 ... Y_k$ é uma regra de produção para algum $k \geq 1$, então acrescente a a $FIRST(X)$ se, para algum i, a estiver em $FIRST(Y_i)$, e λ estiver em todos os $FIRST(Y_1), ..., FIRST(Y_{i-1})$. Se λ está em $FIRST(Y_j)$ para todo $j = 1, 2, ..., k$, então adicione λ a $FIRST(X)$.
3. Se $X \rightarrow \lambda$ é um regra de produção, então acrescente λ a $FIRST(X)$

Conjunto FOLLOW

Para calcular *FOLLOW* de todos os simbolos NÃO-TERMINAIS S de uma gramática, execute as seguintes regras até que nenhum novo símbolo possa ser acrescentado a qualquer conjunto *FOLLOW*.

1. Coloque $\#$ em $FOLLOW(S)$, onde S é o símbolo inicial da gramática e $\#$ é o marcador de fim de sentença, que é incluido antes da avaliação da sentença.
2. Se houver uma produção $A \rightarrow \alpha B \beta$, então tudo que está em $FIRST(\beta)$ exceto λ, deve estar em $FOLLOW(B)$.
3. Se houver uma produção $A \rightarrow \alpha B$ ou $A \rightarrow \alpha B \beta$, onde $FIRST(\beta)$ contém λ, então inclua $FOLLOW(A)$ em $FOLLOW(B)$.

Exemplo

Considere a seguinte gramática:

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' | \lambda \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' | \lambda \\
F &\rightarrow a | (E)
\end{aligned}
$$

Conjuntos FIRST e FOLLOW:

	FIRST	FOLLOW
E	$\{(, a\}$	$\{), \#\}$
E'	$\{+, \lambda\}$	$\{), \#\}$
T	$\{(, a\}$	$\{+,), \#\}$
T'	$\{*, \lambda\}$	$\{+,), \#\}$
F	$\{(, a\}$	$\{*, +,), \#\}$

Algoritmo para Construção da Tabela LL(1)

- Para cada produção $A \rightarrow \alpha$ da gramática, faça:
  1. Para cada terminal a de $FIRST(\alpha)$, adicione a produção $A \rightarrow \alpha$ a $T[A, a]$
  2. Se $FIRST(\alpha)$ inclui a palavra vazia, então adicione $A \rightarrow \alpha$ a $T[A, b]$ para cada b em $FOLLOW(A)$.

	a	$+$	$*$	$($	$)$	$\#$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$E' \rightarrow \lambda$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \lambda$	$T' \rightarrow *FT'$		$T' \rightarrow \lambda$	$T' \rightarrow \lambda$
F	$F \rightarrow a$			$F \rightarrow (E)$		

Definição - Gramática LL(1)

Definição

Uma gramática G não recursiva à esquerda é LL(1) se e somente se, sempre que $A \to \alpha$ *e* $A \to \beta$ *são regras de produção de G, ocorre que:*

1. *a interseção dos conjuntos* $FIRST(\alpha)$ *e* $FIRST(\beta)$ *é vazia;*
2. *no máximo um dos dois,* α *ou* β*, deriva a palavra vazia; e*
3. *se* $\beta \overset{*}{\Rightarrow} \lambda$*, então a interseção de* $FIRST(\alpha)$ *e* $FOLLOW(A)$ *é vazia;*

LR

- Os analisadores ascendentes (*bottom-up*) podem ser construídos com uma classe de gramática chamada LR(1).
  - O primeiro "L"se refere a forma como é lida a cadeia de entrada na análise, nesse caso da esquerda para direita (*Left-to-right*)
  - O segundo "R"se refere a forma como é obtida a sequência de derivação para obtenção da sentença avaliada, nesse caso derivação mais à direita (*rightmost*)
  - O número "1"se refere ao número de símbolos a frente deve-se olhar para decidir que regra da gramática deve ser utilizada.

Algoritmo Análise Ascendente

1. α = cadeia dada
2. Decompor $\alpha = \beta X_1 X_2 ... X_n \gamma$ tal que exista uma regra de produção $X \to X_1 X_2 ... X_n$. Adotar a cadeia $\alpha = \beta X \gamma$, associando-se uma árvore onde X é a raiz e $X_1 X_2 ... X_n$ são as subárvores da raiz X. (Este processo inverso da derivação é denominado REDUÇÃO).
3. O passo 2 é repetido até que o valor de α seja reduzido para o símbolo inicial da gramática.

Implementação

- Na descrição genérica do algoritmo de análise ascendente usa-se a **intuição** para decidir que regra de produção utilizar para redução.
- Para **automatizar** o processo de análise sintática existem alguns algoritmos que utilizam uma tabela (matriz) representando a gramática para decidir de forma automática as reduções.
- A idéia básica deste método é que para cada símbolo da cadeia de entrada é feita uma consulta na tabela. O valor obtido da tabela determina a ação que o algoritmo deve tomar (empilhar um estado, reduzir, aceitar ou rejeitar)
- A consulta na tabela e a execução das ações ocorrem até que a cadeia que está sendo verficada é <u>rejeitada</u> ou <u>aceita</u>.

A tabela de análise LR

A tabela de análise é uma matriz retangular cujas linhas são indexadas pelos estados, e as colunas pelos símbolos do vocabulário da gramática (terminais e não-terminais). Os elementos da matriz indicam as ações que podem ser tomadas pelo algoritmo que podem ser:

- Empilhar o estado e_i
- Reduzir usando a j-ésima regra de produção.
- Aceitar
- Rejeitar

Exemplo de tabela LR

$$
\begin{array}{llll}
(1) & E & \rightarrow & +EE \\
(2) & E & \rightarrow & *EE \\
(3) & E & \rightarrow & a \\
(4) & E & \rightarrow & b
\end{array}
$$

Tabela	E	+	*	a	b	#
e_o	e_1	e_2	e_3	e_4	e_5	
e_1						a
e_2	e_6	e_2	e_3	e_4	e_5	
e_3	e_7	e_2	e_3	e_4	e_5	
e_4		r_3	r_3	r_3	r_3	r_3
e_5		r_4	r_4	r_4	r_4	r_4
e_6	e_8	e_2	e_3	e_4	e_5	
e_7	e_9	e_2	e_3	e_4	e_5	
e_8		r_1	r_1	r_1	r_1	r_1
e_9		r_2	r_2	r_2	r_2	r_2

```
Inicio
P[0] ← e_o; i ← 0; termino ← falso;
reduzido ← falso; Simbolo ← PROXIMO()
Repita
    Se reduzido
        Entao s ← SimboloReduzido
        Senao s ← Simbolo
    FimSe
    Caso Tabela[P[i],s] de
        Empilha (e_j):
            i ← i + 1
            P[i] ← e_j
            Se reduzido
                Entao reduzido ← falso
                Senao Simbolo ← PROXIMO()
        Reduzir (A → α):
            i ← i − | α |
            Reduzido ← verdadeiro
            SimboloReduzido ← A
        Aceitar: termino ← verdadeiro
        Rejeitar: ERRO ( )
    FimCaso
Ate termino
```

Passo	Pilha	Símbolo Reduzido	Cadeia de Entrada	Ação
0	e_o		+*a+baa#	e2
1	$e_o +_2$		*a+baa#	e3
2	$e_o +_2 *_3$		a+baa#	e4
3	$e_o +_2 *_3 a_4$		+baa#	r3
4	$e_o +_2 *_3$	E	+baa#	e7
5	$e_o +_2 *_3 E_7$		+baa#	e2
6	$e_o +_2 *_3 E_7 +_2$		baa#	e5
7	$e_o +_2 *_3 E_7 +_2 b_5$		aa#	r4
8	$e_o +_2 *_3 E_7 +_2$	E	aa#	e6
9	$e_o +_2 *_3 E_7 +_2 E_6$		aa#	e4
10	$e_o +_2 *_3 E_7 +_2 E_6 a_4$		a#	r3
11	$e_o +_2 *_3 E_7 +_2 E_6$	E	a#	e8
12	$e_o +_2 *_3 E_7 +_2 E_6 E_8$		a#	r1
13	$e_o +_2 *_3 E_7$	E	a#	e9
14	$e_o +_2 *_3 E_7 E_9$		a#	r2
15	$e_o +_2$	E	a#	e6
16	$e_o +_2 E_6$		a#	e4
17	$e_o +_2 E_6 a_4$		#	r3
18	$e_o +_2 E_6$	E	#	e8
19	$e_o +_2 E_6 E_8$		#	r1
20	e_o	E	#	e1
21	$e_o E_1$		#	ACEITAR

Construção da Tabela LR

Definições:

- **Item**: É uma regra de produção na qual foi marcada uma posição na cadeia do lado direito; esta posição será indicada por meio do símbolo $\bullet$ (ponto). Exemplo: Seja a gramática:

$$\begin{array}{lcl} E & \rightarrow & +EE \\ E & \rightarrow & *EE \\ E & \rightarrow & a \\ E & \rightarrow & b \end{array}$$

  O conjunto de itens derivados desta gramática é: $\{E \rightarrow \bullet + EE | + \bullet EE | + E \bullet E | + EE \bullet | \bullet * EE | * \bullet EE | * E \bullet E | * EE \bullet | \bullet a | a \bullet | \bullet b | b \bullet\}$. O conjunto de itens para uma gramática é sempre finito e será utilizado para construir os estados da tabela.

Construção da Tabela LR

- **Estado**: É um conjunto de itens. A presença no topo da pilha de um estado contendo um item da forma $A \rightarrow \alpha \bullet \beta$ indica que já foi processada e deslocada para pilha a parte inicial *alpha* do redutendo $\alpha\beta$. O estado contendo o item da forma $A \rightarrow \alpha \bullet$ indica um redutendo completo (item completo), o que indica que a próxima ação será uma REDUÇÃO.

Construção da Tabela LR

- **Fecho**: Diremos que um conjunto K de itens é fechado se para todo item K da forma $A \rightarrow \alpha \bullet B\beta$, todos os itens da forma $B \rightarrow \bullet\gamma$ estão em K. Denotaremos por FECHO(K) o menor conjunto fechado que contém K.

Fecho - exemplo de cálculo

Consideremos os seguintes conjuntos de itens:

$$\begin{aligned} K_1 &= \{E \rightarrow + \bullet EE\} \\ K_2 &= \{E \rightarrow +E \bullet E \mid * \bullet EE \mid \bullet a\} \\ K_3 &= \{E \rightarrow \bullet b\} \end{aligned}$$

Para a gramática:

$$\begin{aligned} E &\rightarrow +EE \\ E &\rightarrow *EE \\ E &\rightarrow a \\ E &\rightarrow b \end{aligned}$$

Então seus fechos são:

$$\begin{aligned} FECHO(K_1) &= \{E \rightarrow + \bullet EE \mid \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b\} \\ FECHO(K_2) &= \{E \rightarrow +E \bullet E \mid * \bullet EE \mid \bullet a \mid \bullet +EE \mid \bullet *EE \mid \bullet b\} \\ FECHO(K_3) &= \{E \rightarrow \bullet b\} \end{aligned}$$

FECHO (closure)

FECHO (K)

Pergunta: No conjunto de itens que forma K, tem algum item em que o ponto precede algum símbolo não-terminal X? Se sim, acrescente todos as regras desse não terminal X, como item para formar o conjunto fechado de K.

Construção da Tabela LR

- **Transfere**: Vamos agora analisar como determinar as entradas da forma e_j na tabela de análise. Suponhamos que um estado e_i contenha um item incompleto da forma $A \to \alpha \bullet X\beta$. A presença deste estado no topo da pilha de análise indica que se o próximo símbolo a ser consultado for X, então terá sido processada a parte αX do redutendo $\alpha X\beta$, devendo ser empilhado portando um estado que contenha o item $A \to \alpha X \bullet \beta$ (com a marca depois do símbolo X).
- Definiremos então a função TRANSFERE (K,X), como sendo o conjunto fechado de todos os itens da forma $A \to \alpha X \bullet \beta$ tais que o item $A \to \alpha \bullet X\beta$ está em K.

Transfere - exemplo de cálculo

Consideremos a gramática anterior e os conjuntos:

$$
\begin{aligned}
K_1 &= \{E \to * \bullet EE\} \\
K_2 &= FECHO(K_1) = \{E \to + \bullet EE | \bullet +EE | \bullet *EE | \bullet a | \bullet b\} \\
K_3 &= \{E \to \bullet b\}
\end{aligned}
$$

Tem-se então:

$$
\begin{aligned}
TRANSFERE(K_1, *) &= \{\} \\
TRANSFERE(K_1, E) &= \{E \to *E \bullet E | \bullet +EE | \bullet *EE | \bullet a | \bullet b\} \\
TRANSFERE(K_2, +) &= \{E \to + \bullet EE | \bullet +EE | \bullet *EE | \bullet a | \bullet b\} = K_2 \\
TRANSFERE(K_2, a) &= \{E \to a\bullet\} \\
TRANSFERE(K_3, E) &= \{\} \\
TRANSFERE(K_3, b) &= \{E \to b\bullet\}
\end{aligned}
$$

TRANSFERE (goto)

TRANSFERE (K, X)

Pergunta: No conjunto de itens que forma o estado K, tem algum item em que o ponto precede o símbolo X? Se sim, transfere o ponto para depois de X nesse(es) item(ns) e calcula o FECHO.

Algoritmo para Cálculo da coleção de estados

1. Adota-se o estado $e_0 = FECHO(\{S' \to \bullet S\#\})$ como sendo o valor inicial da coleção C. Observe que deve ser acrescentada à gramática, uma regra para caracterizar o instante que a sentença toda será reduzida para o símbolo inicial. O símbolo terminal $\#$ é artificialmente acrescentado a gramática para marcar o fim da sentença que será analisada.
2. Se existe um estado e de C e um símbolo X de Σ (vocabulário) tais que $e' = TRANSFERE(e, X) \neq \emptyset$ e $e' \notin C$, então e' é acrescentado à coleção C.
3. O passo 2 é repetido até que não se possam acrescentar mais estados à coleção C.

C é o conjunto de estado tipo LR(0) da gramática.

Exemplo de cálculos

$$\begin{array}{llll} (0) & S' & \rightarrow & S\# \\ (1) & S & \rightarrow & aS \\ (2) & S & \rightarrow & b \end{array}$$

$$\begin{aligned}
e_0 &= FECHO(\{S' \rightarrow \bullet S\#\}) = \{S' \rightarrow \bullet S\# \\
&\qquad\qquad S \rightarrow \bullet aS| \bullet b\} \\
e_1 &= TRANSFERE(e_0, S) = \{S' \rightarrow S \bullet \#\} \\
e_2 &= TRANSFERE(e_0, a) = \{S \rightarrow a \bullet S| \bullet aS| \bullet b\} \\
e_3 &= TRANSFERE(e_0, b) = \{S \rightarrow b\bullet\} \\
e_4 &= TRANSFERE(e_2, S) = \{S \rightarrow aS\bullet\} \\
&\quad TRANSFERE(e_2, a) = e_2 \\
&\quad TRANSFERE(e_2, b) = e_3
\end{aligned}$$

	S	a	b	#
e_o	e_1	e_2	e_3	
e_1				a
e_2	e_4	e_2	e_3	
e_3		r_2	r_2	r_2
e_4		r_1	r_1	r_1

$$C = \{e_0, e_1, e_2, e_3, e_4\}$$