Gramáticas

(parte “b”)

Eduardo Furlan Miranda

Baseado em: GARCIA, A. de V.; HAEUSLER, E. H. Linguagens Formais e Autômatos. Londrina: EDA, 2017.

Fecho transitivo-reflexivo

Transitividade: um ou mais passos ($\Rightarrow^+$)
Reflexividade: zero ou mais passos ($\Rightarrow^*$)

- $\Rightarrow_G{}^*$ significa mais de um passo da relação $\Rightarrow_G$
  - $\Rightarrow_G{}^*$ é o fecho transitivo-reflexivo de $\Rightarrow_G$

$\Rightarrow$ representa uma única derivação

- Dado $N \Rightarrow_G L.L \Rightarrow_G D.L \Rightarrow_G 1.L$, podemos escrever $N \Rightarrow_G{}^* 1.L$
  - O $*$ indica que $\Rightarrow_G{}^*$ também é reflexiva, ou seja, $\alpha \Rightarrow^* \alpha$
    - Reflexividade: qualquer símbolo pode ser derivado em si mesmo em zero passos, sem realizar nenhuma substituição
      - $\Rightarrow^*$ representa "zero ou mais passos de derivação"

- Esta notação é lida como "deriva em zero ou mais passos"
  - $N \Rightarrow_G{}^* 1.L$ é pronunciado "N deriva em zero ou mais passos 1.L "

$\subseteq$: símbolo de subconjunto

- Definição formal do fecho transitivo-reflexivo da relação de derivação $\Rightarrow_G^*$ em uma gramática formal G
  - Dada uma gramática G(V, T, P, S), dizemos que a relação $\Rightarrow_G^* \subseteq (V \cup T)^* \times (V \cup T)^*$ é a menor relação que satisfaz 3 condições:
    - Se $\alpha \Rightarrow_G \beta$ então $\alpha \Rightarrow_G^* \beta$
      - A relação $\Rightarrow_G$ está contida em $\Rightarrow_G^*$
    - Para todo $\omega \in (V \cup T)^*$, $\omega \Rightarrow_G^* \omega$
      - Lê-se: para toda cadeia ω que pertence ao fecho de Kleene do conjunto união de variáveis (V) e terminais (T), ω pode ser derivada em si mesma através da relação $\Rightarrow_G^*$, em zero passos de derivação

ω omega
V variáveis
T terminais
* "vazia ou mais" cadeias

(continua)

(continuação)

- Para todos $\alpha, \beta, \gamma \in (V \cup T)^*$, se $\alpha \Rightarrow_G^* \beta$ e $\beta \Rightarrow_G^* \gamma$, então $\alpha \Rightarrow_G^* \gamma$
  - Propriedade de transitividade: se α pode ser derivado em β em zero ou mais passos, e β pode ser derivado em γ em zero ou mais passos, então α pode ser derivado em γ em zero ou mais passos
    - Isso significa que podemos "encadear" derivações

Representação de uma gramática

- Considere a gramática G = (V , T , P , S) onde:
  - V = { S, B }
  - T = { a }
  - P = { $S \rightarrow aB$, $B \rightarrow \varepsilon$ }

- Esta gramática gera uma única sentença "a"
  - Pode ser obtida pela derivação $S \Rightarrow aB \Rightarrow a$, com $B \rightarrow \varepsilon$

- Podemos definir a gramática anterior em duas linhas simples:
  - $S \rightarrow aB$
  - $B \rightarrow \varepsilon$
    - Variáveis serão sempre letras maiúsculas
    - Os demais símbolos serão terminais
    - O símbolo inicial fica do lado esquerdo da primeira regra

- Definição formal de linguagem L(G) gerada por uma gramática G = (V, T, P, S) :
  - Conjunto de todas as sentenças $\alpha \in T^*$ tais que $S \Rightarrow_G^* \alpha$, onde
    - T^* : conjunto de todas as cadeias possíveis que podem ser formadas usando os símbolos terminais T, incluindo a cadeia vazia ε
      - $T^* = \{\varepsilon, a, aa, aaa, \ldots\}$
    - $\Rightarrow_G^*$: relação de derivação em zero ou mais passos na gramática G (fecho transitivo-reflexivo)
    - $\alpha \in T^*$: α é uma string composta apenas por:
      - símbolos terminais
      - string vazia

(continua)

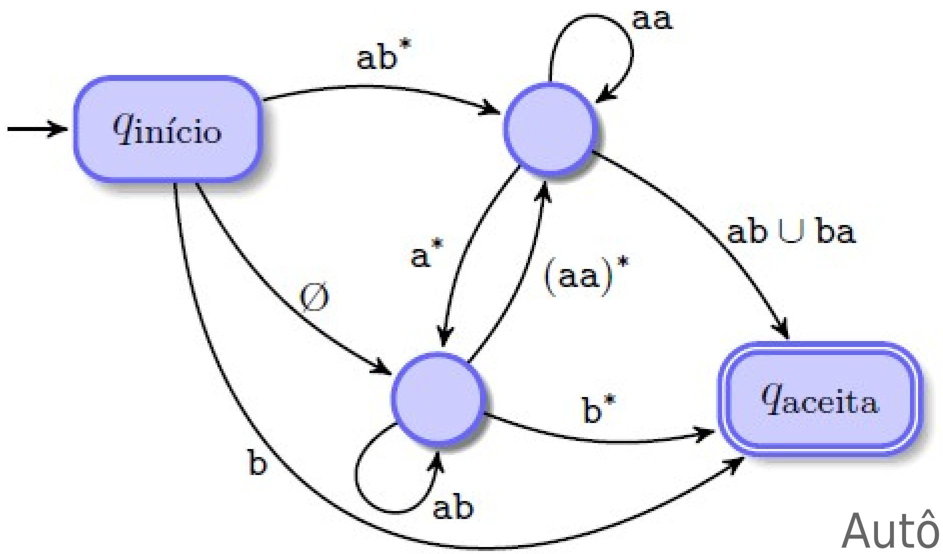
- L(G) : linguagem gerada por G
  - Conjunto de todas as strings α que podem ser derivadas a partir do símbolo inicial S em zero ou mais passos
    - usando as regras de produção em P, e que contêm apenas símbolos terminais
- O resultado final, se existir, deve ser uma string composta exclusivamente por símbolos terminais
- Todas as strings que podem ser obtidas dessa forma pertencem à linguagem gerada pela gramática

- Dada uma gramática G (V , T , P , S) dizemos que a linguagem gerada por G é $L(G) = \{ \omega \in T^* \mid S \Rightarrow_G^* \omega \}$
  - $\omega = \omega$ (ômega) é uma string que pertence ao conjunto T^* composta exclusivamente por símbolos terminais
  - $S \Rightarrow_G^* \omega$: o símbolo inicial S pode ser derivado em ω em zero ou mais passos, usando as regras de produção da gramática G

- A linguagem gerada por G é o conjunto de todas as strings ω compostas exclusivamente por símbolos terminais, que podem ser derivadas a partir do símbolo inicial S, aplicando as regras de produção da gramática G, zero ou mais vezes

Hierarquia de Chomsky (1956)

- Afirma que as linguagens formam uma hierarquia a partir dos tipos das gramáticas que são capazes de gerá-las
- Cada tipo gramatical também nos indicará o mecanismo ou autômato capaz de resolver o problema da análise sintática para aquele tipo de gramática



Autômato

Tipos de regras gramaticais

- Regular $\{ \varepsilon, b, ab, aab, aaab, \ldots \}$, $a^n b$
  - $A \rightarrow b$, $A \rightarrow \varepsilon$, $A \rightarrow aB$ ou $A \rightarrow Ba$ (as regras mais restritas)
- Livre de Contexto
  - $A \rightarrow \gamma$ (menos restritas) $\{ ab, aabb, aaabbb, aaaabbbb, \ldots \}$, $a^n b^n$
    - A é uma variável
    - γ é uma sequência de variáveis e/ou terminais
- Sensível ao Contexto $\{ abc, aabbcc, aaabbbccc, \ldots \}$, $a^n b^n c^n$
  - $\alpha \rightarrow \beta$
    - α e β são sequências de variáveis e/ou terminais
    - o comprimento de α é menor ou igual ao de β
- Irrestrita (ou Geral) $\{ a, aa, aaaa, aaaaaaaa, \ldots \}$, a^{2^n}
  - pelo menos uma variável no lado esquerdo da regra

Hierarquia de Chomsky

R	$a^n b$	ε, b, ab, aab
LC	$a^n b^n$	ab, aabb
SC	$a^n b^n c^n$	abc, aabbcc
I	a^{2^n}	a, aa, aaaa

apenas estas

Gramáticas	Regras	Ex. de linguagens geradas
GR (tipo 3) Regulares	$A \to aB$, $A \to b$, ($A \to \varepsilon$, se permitido, apenas para o símbolo inicial) $A, B \in V$ (variáveis) $a, b \in T$ (terminais)	{ ε, b, ab, aab, aaab, ... } $= \{ a^n b \mid n \geq 0\} \cup \{\varepsilon\}$
GLC (tipo 2) Livres de Contexto	$A \to \alpha$ $A \in V$, $\alpha \in (V \cup T)^*$ A: 1 única variável	{ ab, aabb, aaabbb, aaaabbbb, ... } $= \{ a^n b^n \mid n > 0 \}$
GSC (tipo 1) Sensíveis ao Contexto	$\alpha \to \beta$ $\alpha \in (V \cup T)^+$, $\beta \in (V \cup T)^*$ $\mid \alpha \mid \leq \mid \beta \mid$ $S \to \varepsilon$, se S não aparece do lado direito de nenhuma regra	{ abc, aabbcc, aaabbbccc, ... } $= \{ a^n b^n c^n \mid n > 0 \}$
GI (tipo 0) Irrestrita ou geral	$\alpha \to \beta$ $\alpha, \beta \in (V \cup T)^*$ α: pelo menos 1 símbolo de V	{ a, aa, aaaa, aaaaaaaa, ... } $= \{ a^{2^n} \mid n \geq 0\}$

GF e LF

(continuação da tabela anterior)

Hierarquia Chomsky	**Gramática**	**Linguagem**	**Reconhecedor**
Tipo 3	Regular	Regular	Autômato finito
Tipo 2	Livre de Contexto	Livre de Contexto	Autômato com pilha
Tipo 1	Sensível ao Contexto	Sensível ao Contexto	Autômato linearmente limitado
Tipo 0	Irrestrita	Recursivamente enumerável	Máquina de Turing
		Recursiva	Máquina de Turing que sempre para

Hierarquia de regras e linguagens formais

contido

- Regular ⊂ Livre de Contexto ⊂ Sensível ao Contexto ⊂ Irrestrita
  - Ex.: toda regra regular é livre de contexto
    - Nem toda regra livre de contexto é regular
- Linguagens herdam o tipo da gramática mais restrita que as gera
- Se a gramática não é de um tipo específico
  - Significa que a linguagem não pertence a esse tipo

Hierarquia de Chomsky e Poder Expressivo

- Gramáticas Regulares (GR)
  - As mais restritas
  - Usadas para descrever linguagens regulares, reconhecidas por autômatos finitos
  - São eficientes para análise léxica (quebrar o código fonte em tokens)
    - Mas não conseguem expressar estruturas sintáticas complexas

- Gramáticas Livres de Contexto (GLC)
  - Um meio-termo entre poder expressivo e eficiência
  - Conseguem expressar a maioria das estruturas sintáticas das linguagens de programação
  - Existem algoritmos eficientes para análise (parsing)
  - Tem o poder expressivo necessário para gerar a linguagem $\{a^n b^n \mid n>0\}$ ({ ab, aabb, aaabbb, aaaabbbb, ... })
  - Gramáticas regulares não possuem essa capacidade

Hierarquia de Chomsky e Poder Expressivo

- Gramáticas Sensíveis ao Contexto (GSC)
  - Mais poderosas que as GLCs
  - Mas com algoritmos de análise muito mais complexos e custosos
  - Raramente usadas em compiladores práticos
  - Tem o poder computacional necessário para expressar dependências entre múltiplos símbolos
  - GLCs e gramáticas regulares não possuem essa capacidade

Hierarquia de Chomsky e Poder Expressivo

- Gramáticas Irrestritas (GI)
  - As mais poderosas, equivalentes a Máquinas de Turing (MT)
  - Capazes de expressar qualquer linguagem recursivamente enumerável, mas com algoritmos de análise indecidíveis em geral (não há garantia de que um algoritmo de análise sempre termine)

Modelo abstrato de computador. Se temos um conjunto de regras de produção em uma GI que define uma determinada linguagem, sempre podemos construir uma MT que reconheça todas as strings pertencentes a essa linguagem e rejeite as strings que não pertencem

Pode enumerar todas as cadeias válidas da linguagem, mas não é garantido que a MT pare para cadeias inválidas

Exemplo de gramática regular

- Gramáticas regulares (GR) são um subconjunto das gramáticas livres de contexto (GLC)

- A gramática com as regras $A \to aA$ e $A \to b$ gera a linguagem $a^n b$
  - onde n é um número inteiro maior que 0
  - $A \to aA$ gera zero ou mais ocorrências de a ("a^n"), pois A é recursivo
  - $A \to b$ substitui A por b, terminando a recursão

- A derivação sempre mantém apenas uma variável, e ela está sempre no final da cadeia
  - $A \Rightarrow aA$
  - $A \Rightarrow aaA$
  - $A \Rightarrow aab$ aplicando a regra $A \to b$

Regra livre de contexto

- Regra Livre de Contexto
  - Em uma regra livre de contexto $A \rightarrow \gamma$, a variável A pode ser substituída por γ independentemente do contexto em que A ocorre

- Algumas formas de regras livres de contexto
  - $A \rightarrow aB$: uma variável A se transforma em um terminal a seguido de outra variável B
  - $A \rightarrow \varepsilon$: uma variável A se transforma na cadeia vazia
  - $A \rightarrow b$: uma variável A se transforma em um terminal b
    - A e B : representam variáveis quaisquer
    - a e b : representam terminais quaisquer

Regra sensível ao contexto

- Uma regra é sensível ao contexto se tem a forma $uA\upsilon \rightarrow u\gamma\upsilon$
  - A é uma variável
  - u e υ são cadeias de símbolos (o contexto)
  - γ é uma cadeia não vazia

- Algumas formas de regras sensíveis ao contexto
  - $aAa \rightarrow abba$: uma variável A é substituída por bb apenas quando está cercado por a
  - $0A1 \rightarrow 01$: uma variável A é substituída por 1 apenas quando está entre 0 e 1
  - $abcAd \rightarrow abcd$: A é substituído por d apenas quando está após abc

Gramática não regular, e linguagem regular

- Seja a gramática G_1 : $A \to aA$, $A \to b$
  - G_1 só possui regras regulares: é uma gramática regular
    - Gera $L = \{ a^k b , k > 0 \}$

GR é só $A \to aB$, $A \to b$, $A \to \varepsilon$

- Seja a gramática G_2 : $A \to aaA$, $A \to aA$, $A \to b$
  - G_2 possui regra que não é regular, e portanto a gramática G_2 não é regular

- G_1 e G_2 geram a mesma linguagem (b, ab, aab, ...)

- Apesar de G_2 não ser uma gramática regular (devido $A \to aaA$), a linguagem que ela gera ($L(G_2)$) é uma linguagem regular porque existe uma gramática regular (G_1) que gera a mesma linguagem

- Geralmente, os compiladores usam uma forma de gramática livre de contexto (GLC), ou mais especificamente, subclasses de GLCs que permitem uma análise sintática eficiente

- A razão principal para essa escolha é o equilíbrio que as GLCs oferecem entre poder expressivo e eficiência de análise

- Vantagens das GLCs
  - Equilíbrio entre poder expressivo e eficiência
  - Facilitam a análise sintática
  - Permitem a definição de estruturas complexas
  - São adequadas para linguagens de programação

Análise sintática

número de símbolos de lookahead que o analisador sintático considera para decidir qual regra de produção aplicar

- Subclasses de GLCs utilizadas
  - Gramáticas LL(1) (Left-to-Right, Leftmost derivation)
  - Gramáticas LR(1) (Left-to-Right, Rightmost derivation)
  - Gramáticas LALR(1) (Look-Ahead Left-to-Right) (variação de LR(1))

regras de produção começam no símbolo mais à esquerda

lê os símbolos da entrada na ordem que aparecem

- Razões para escolha
  - Eficiência na análise sintática
  - Facilidade de implementação
  - Capacidade de lidar com linguagens complexas
- Exemplos de uso
  - Compiladores de linguagens como C, C++ e Java
  - Analisadores sintáticos
  - Ferramentas de desenvolvimento