Aula 03 – Hierarquia de Chomsky

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação

Introdução

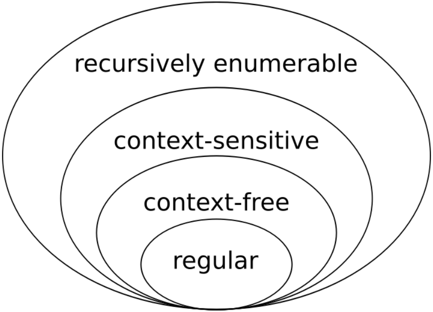
- Aula 02: máquinas de Turing como máquinas para reconhecer uma linguagem recursivamente enumerável.
- Revisão de LFA: autômatos finitos como máquinas para reconhecer linguagens regulares.
- **Estes *slides***: relação entre expressividade das linguagens e poder de reconhecimento das máquinas.
- **Objetivos**: apresentar os níveis de linguagens e máquinas que formam a Hierarquia de Chomsky.

Referências

Section 7.1 & Chapter 10 – The Chomsky Hierarchy
T. Sudkamp
Chapter 2 – Context-Free Languages
M. Sipser
Chapter 3 – Context-Free Languages
A. Maheshwari

Hierarquia de Chomsky

- Proposta pelo linguista Noam Chomsky em 1956.
- Condensa todos os resultados de linguagens formais, utilizadas em computação e em linguística.
- A Hierarquia de Chomsky (HC) é uma hierarquia de inclusão de classes de gramáticas formais.
- A HC é formada por quatro tipos de linguagens, numeradas de 3 a 0. Linguagens regulares são do tipo 3.



Máquinas Abstratas e HC

- Cada tipo de linguagem da HC possui uma máquina associada que reconhece esse tipo.
- Além dos autômatos finitos (AFs) e máquinas de Turing (TMs), há duas outras máquinas, listadas a seguir.
- Autômato de Pilha (*Pushdown Automaton* – PDA), um AF com memória FILO.
- Autômato Linear (*Linear-bounded Automaton* – LBA), uma TM com fita limitada.
- A HC é resumida na tabela adiante.
- Mas antes precisamos definir o conceito de uma gramática formal.

Gramáticas Formais

Definição – Gramática Formal

Uma gramática G é uma tupla $G = (V, \Sigma, P, S)$, onde:

- V é um conjunto finito de variáveis (não-terminais);
- Σ (o alfabeto) é um conjunto finito de símbolos terminais;
- P é um conjunto finito de regras (produções); e
- $S \in V$ é símbolo inicial de G.

- Os conjuntos V e Σ devem ser disjuntos.
- Aplicação da regra $u \to v$ na string xuy produz xvy.
- A forma $xuy \Rightarrow xvy$ é dita uma derivação.
- A forma $p \Rightarrow^* q$ indica que q é derivável de p por zero ou mais aplicações de regras.
- A linguagem de uma gramática G é o conjunto de strings do alfabeto deriváveis do símbolo inicial S.
- Simbolicamente, $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Hierarquia de Chomsky

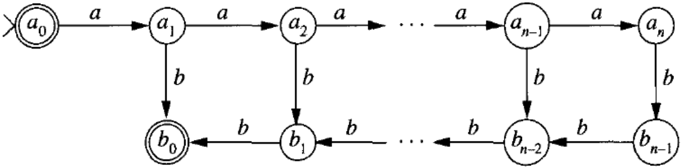
Chomsky Type	Language Type	Grammar Restriction	Automaton Type (Memory?)
Type 3	Regular	LHS Single variable RHS either (a) a single word, e.g. $S \rightarrow x$ (b) or, a single word plus single variable e.g. $S \rightarrow aX$	Finite state automata (No memory)
Type 2	Context Free	LHS single variable RHS can be anything e.g. $S \rightarrow aXYbh$	Pushdown stack automata (Stack memory)
Type 1	Context Sensitive	e.g. $aZ \rightarrow Y$ i.e "Z goes to Y provided a is on the left" RHS never shorter than LHS	Linear bounded Automata (Bounded RAM)
Type 0	Recursive	No restrictions but productions terminate	Turing machines (Unlimited RAM)
Type 0	Recursively Enumerable	No restrictions but may loop for ever	Turing machines (Unlimited RAM)

Tipo 3 – Linguagens Regulares e Autômatos Finitos

- Linguagens regulares são o tipo mais simples de linguagem formal.
- Compostas somente por três operações fundamentais: concatenação, união e fecho (estrela) de Kleene (*).
- São descritas por expressões regulares mas também podem ser representadas por gramáticas formais.
- Restrições para as regras ($u \rightarrow v$):
  1. u é somente um não-terminal, isto é, $u \in V$.
  2. v é somente um terminal ($X \rightarrow a$), ou um terminal à esquerda ou direita de um não-terminal ($X \rightarrow aA$ ou $X \rightarrow Aa$).
- Condição 2 acima permite uma forma limitada de recursão à esquerda ou à direita: $X \rightarrow Xa$ ou $X \rightarrow aX$.

Tipo 3 – Linguagens Regulares e Autômatos Finitos

- Linguagens regulares são reconhecidas por AFs.
- A linguagem $L_3 = \{a^i b^i \mid i \leq n\}$, para um natural fixo n, é regular.
- AF que reconhece L_3:



- A linguagem $L_2 = \{a^i b^i \mid i \geq 0\}$ não é regular.
- Exigiria um AF com "infinitos" estados para contar i.
- AFs têm uma "memória" muito limitada: somente o estado atual.

Tipo 2 – Linguagens Livre de Contexto e PDAs

- Linguagens livres de contexto (*context-free languages* – CFL) são geradas por gramáticas livres de contexto (*context-free grammars* – CFG).
- Restrições para as regras ($u \to v$):
  1. u é somente um não-terminal, isto é, $u \in V$.
  2. v pode ser qualquer combinação de terminais e não-terminais, isto é, $v \in (V \cup \Sigma)^*$.
- CFLs são reconhecidas por Autômatos de Pilha (PDAs).
- Um PDA é um AF aumentado com uma memória FILO, isto é, uma pilha.

Tipo 2 – Linguagens Livre de Contexto e PDAs

Definição 7.1.1 (Sudkamp) – *Pushdown automaton (PDA)*

Um PDA é uma tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ onde:

- Q é um conjunto finito de estados.
- Σ é o alfabeto de entrada.
- Γ é o alfabeto da pilha.
- $q_0 \in Q$ é o estado inicial.
- $F \subseteq Q$ é o conjunto de estados finais.
- $\delta\colon Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \rightharpoonup \mathcal{P}(Q \times (\Gamma \cup \{\lambda\}))$ é a função de transição.

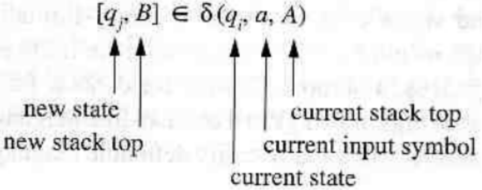
- Os alfabetos de um PDA são disjuntos.
- O alfabeto Σ é usado para construir a string de entrada.
- O alfabeto Γ indica os símbolos que podem ser colocados/removidos na pilha.

Tipo 2 – Linguagens Livre de Contexto e PDAs

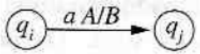
- Como o co-domínio da função de transição é descrito como um conjunto potência $\mathcal{P}$, sabemos que PDAs são máquinas não-determinísticas.
- A pilha é representada como uma string de Γ^*.
- O elemento mais à esquerda da string é o topo da pilha.
- Notação: a pilha $A\alpha$ tem o símbolo $A \in \Gamma$ como o topo e a substring $\alpha \in \Gamma^*$ como o resto da pilha. A pilha vazia é denotada por λ.
- A computação do PDA começa com a máquina no estado q_0, com a entrada na fita e a pilha vazia.

Tipo 2 – Linguagens Livre de Contexto e PDAs

- Uma transição da máquina lê o símbolo atual da entrada e o topo da pilha, como abaixo.

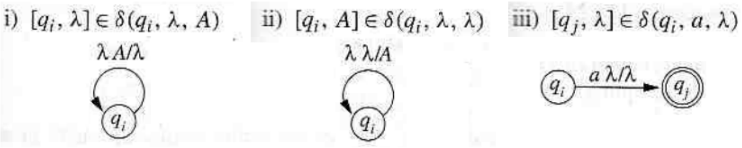


- Ao tomar a transição acima, a máquina:
  - Passa do estado q_i para o estado q_j.
  - Processa o símbolo a (move a cabeça de leitura uma posição para a direita).
  - Desempilha A do topo da pilha (*pops A*).
  - Empilha B no topo da pilha (*pushes B*).
- A transição acima pode ser ilustrada como abaixo.



Tipo 2 – Linguagens Livre de Contexto e PDAs

- Um argumento λ para a função δ indica que ele não deve ser considerado para a computação da transição.
- A figura abaixo mostra os possíveis casos.



i) Desempilha *A* (sem mexer na entrada).
ii) Empilha *A* (sem mexer na entrada).
iii) Processa um símbolo da entrada sem mexer na pilha (igual AF).

Tipo 2 – Linguagens Livre de Contexto e PDAs

Definição 7.1.2 – Critério de aceite e linguagem de um PDA

Seja M um PDA. A string $w \in \Sigma^*$ é aceita por M se existe uma computação

$$[q_0, w, \lambda] \overset{*}{\vdash} [q_f, \lambda, \lambda]$$

onde $q_f \in$ F. A linguagem de M, denotada L(M), é o conjunto de strings aceitas por M.

É essencial perceber que a definição acima coloca três condições para o aceite:

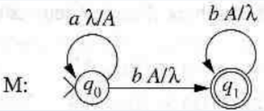
1. A máquina deve parar em um estado final q_f.
2. A entrada deve ser totalmente consumida.
3. A pilha deve estar vazia.

Se uma ou mais dessas condições for violada $\Rightarrow$ rejeita.

Tipo 2 – Linguagens Livre de Contexto e PDAs

Exemplo 7.1.2 (Sudkamp)

O PDA M abaixo aceita a linguagem $L_2 = \{a^i b^i \mid i \geq 0\}$.



Portanto, L_2 é uma linguagem livre de contexto (CFL).

Tipo 1 – Linguagens Sensíveis ao Contexto e LBAs

- Linguagens sensíveis ao contexto (*context-sensitive languages* – CSL) são geradas por gramáticas sensíveis ao contexto (*context-sensitive grammars* – CSG).
- Restrições para as regras ($u \to v$):
  1. $length(u) \leq length(v)$.
- Regras que satisfazem a condição acima são ditas monotônicas.
- Todas as CSLs são recursivas $\Rightarrow$ são decididas por TMs.
- CSLs são reconhecidas por Autômatos Lineares (LBAs).
- Importante entender a diferença entre os itens anteriores. (Veja Aula 02, slides 5 e 6.)

Tipo 1 – Linguagens Sensíveis ao Contexto e LBAs

- A gramática G abaixo é sensível ao contexto (CSG).

$$\begin{aligned} S &\to aAbc \mid abc \\ A &\to aAbC \mid abC \\ Cb &\to bC \\ Cc &\to cc \end{aligned}$$

- A linguagem L(G) da gramática é $L_1 = \{a^i b^i c^i \mid i > 0\}$.
- Portanto, L_1 é uma CSL, e logo, recursiva.
- Uma TM que decide $L_1 \cup \{\lambda\}$ é apresentada no slide 8 da Aula 02 (Exemplo 8.2.2).
- CFLs são aceitas (reconhecidas) por LBAs.

Tipo 1 – Linguagens Sensíveis ao Contexto e LBAs

Definição 10.3.1 – *Linear-bounded automaton* (LBA)

Um LBA é uma tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, \langle, \rangle, F)$, aonde $\langle$ e $\rangle$ são elementos destacados de Σ e os demais componentes são idênticos à uma NTM.

- A configuração inicial é $q_0\langle w\rangle$, exigindo uma fita com $length(w) + 2$ posições.
- Toda computação deve permanecer entre os limites indicados por $\langle$ e $\rangle$.
- Limitar a memória de uma TM diminui o seu poder de computação.

Teorema 10.3.3 (Sudkamp)

Uma linguagem L é aceita por um LBA se, e somente se, L é sensível ao contexto.

Tipo 0 – Ling. Recursivamente Enumeráveis e TMs

- Linguagens recursivas (*recursive*) são geradas por gramáticas irrestritas (*unrestricted grammars*).
- Uma produção de uma gramática irrestrita tem a forma $u \to v$, onde $u \in (V \cup \Sigma)^+$ e $v \in (V \cup \Sigma)^*$.
- A gramática é dita irrestrita porque não há nenhuma limitação sobre as produções, exceto a acima.
- Mas, para a linguagem ser recursiva, é preciso que todas as possíveis derivações sempre terminem.
- Caso contrário, a linguagem é dita recursivamente enumerável (*recursively enumerable*).

Tipo 0 – Ling. Recursivamente Enumeráveis e TMs

Exemplo 10.1.1 (Sudkamp)

Uma gramática irrestrita que gera $L_1 = \{a^i b^i c^i \mid i > 0\}$.

$$
\begin{aligned}
V &= \{S, A, C\} \\
\Sigma &= \{a, b, c\} \\
S &\to aAbc \\
A &\to aAbC \mid \lambda \\
Cb &\to bC \\
Cc &\to cc
\end{aligned}
$$

Derivação para $i = 2$:

$$S \Rightarrow aAbc \Rightarrow aaAbCbc \Rightarrow aabCbc \Rightarrow aabbCc \Rightarrow aabbcc.$$

Obs.: Embora L_1 tenha sido gerada por uma gramática irrestrita, a linguagem na verdade é uma CSL. (Slide 17.)

- Gramáticas irrestritas provêm o tipo mais flexível de transformação de strings.
- É razoável esperar que linguagens de gramáticas irrestritas só possam ser reconhecidas pelo tipo mais poderoso de máquina abstrata.
- Os teoremas abaixo confirmam essa expectativa.

Teorema 10.1.2 (Sudkamp)

Seja G uma gramática irrestrita. Então L(G) é uma linguagem recursivamente enumerável.

Teorema 8.8.6 (adaptado)

Uma linguagem L é aceita por uma TM se, e somente se, L é recursivamente enumerável.

Tipo 0 – Ling. Recursivamente Enumeráveis e TMs

- É um pouco mais difícil de encontrar exemplos de linguagens recursivas e recursivamente enumeráveis.
- Exemplo de uma linguagem recursiva: o conjunto de fórmulas válidas na aritmética de Presburger. (É recursiva mas não é uma CSL.)
- Aritmética de Presburger é similar à aritmética de Peano mas só admite a operação de soma.
- Comentário: Aritmética de Peano é indecidível!
- Exemplo de uma linguagem recursivamente enumerável: o conjunto de todas as TMs que param (i.e., nunca entram em *loop* para qualquer entrada).
- Se removermos a restrição de parada, a linguagem acima fica indecidível.
- Essa é a linguagem do Problema da Parada, que será vista na próxima aula.

Aula 03 – Hierarquia de Chomsky

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação