
INTRODUÇÃO À TEORIA DE AUTÔMATOS
E LINGUAGENS FORMAIS

Edson Luiz França Senne

SOMÁRIO

1	INTRODUÇÃO	1
2	AUTÔMATOS FINITOS E LINGUAGENS REGULARES	12
3	LINGUAGENS LIVRES DE CONTEXTO	51
4	MAQUINAS DE TURING	108
5	LINGUAGENS SENSÍVEIS AO CONTEXTO	134
6	DECIDIBILIDADE	143
	BIBLIOGRAFIA	161

PREFÁCIO

Este trabalho é resultado da experiência do autor em ministrar o curso CAP350 - Autômatos e Linguagens Formais do programa de pós-graduação em computação aplicada do INPE. Seu objetivo principal é servir como material de apoio ao professor e por isto apresenta o assunto de forma que possa ser acomodado num período de 40 horas. Sua leitura pode também ser útil àqueles que desejarem ter uma noção rápida sobre seus aspectos formais da Ciência da Computação. Não é recomendada sua utilização por alunos e por outras pessoas que desejarem ter conhecimentos mais aprofundados sobre autômatos e linguagens formais. Para estas pessoas, sua leitura, embora não deletéria, deverá ser sempre acompanhada (quando não completamente substituída) pela de textos mais elaborados como os que são listados na bibliografia.

O texto está mais voltado para a teoria de linguagens e a apresenta segundo classes cada vez mais abrangentes conforme a hierarquia de Chomsky. Os autômatos são encarados como procedimentos reconhecedores de linguagem e são apresentados juntamente com as gramáticas que geram cada uma dessas classes. Dessa forma, no capítulo 1 apresenta-se a hierarquia de Chomsky e seus mecanismos geradores e reconhecedores. O capítulo 2 dedica-se à classe de linguagens regulares e o capítulo 3 à classe de linguagens livres de contexto. A estes dois capítulos é dada grande atenção devido à sua importância em relação a Linguagens de Programação e Construção de Compiladores. No capítulo 4 abordam-se as máquinas de Turing (e aí, por motivos didáticos, há uma inversão com o capítulo 5 em relação à hierarquia de Chomsky),

discutindo os aspectos importantes desse modelo que embora simples, parece apresentar as condições necessárias e suficientes para a computação em geral. O capítulo 5 é dedicado às linguagens sensíveis ao contexto e por fim, no capítulo 6, discutem-se problemas de decisão, mostrando alguns problemas para os quais não existe um procedimento capaz de resolvê-los completamente, uma questão importante e que diz respeito aos limites teóricos da computação.

São José dos Campos, agosto de 1984

E.L.F.S.

CAPÍTULO 1 - INTRODUÇÃO

1.1 - HISTÓRICO

A Ciência da Computação pode ser vista como constituída de dois grandes componentes:

- (a) as ideias fundamentais e os modelos que suportam a computação;
- (b) as técnicas de engenharia para a construção de sistemas de computação, tanto de "hardware" como de "software".

Este trabalho é uma introdução ao estudo do primeiro componente: as ideias fundamentais que servem de base para a computação.

Muitos trabalhos foram importantes para o desenvolvimento teórico da Ciência da Computação. Nesta seção apresentam-se os trabalhos mais significativos para o estudo de autómatos e linguagens formais, segundo uma perspectiva histórica.

O matemático David Hilbert acreditava que em matemática não existe IGNORABILITAS, ou seja, que todo problema matemático precisamente especificado podia ser resolvido (possivelmente provando a impossibilidade de sua solução). A ideia de Hilbert para a solução de problemas matemáticos era descobrir um procedimento mecânico (efetivo) para verificar a veracidade ou correção das proposições matemáticas. Hilbert chamou de ENTSCHEIDUNGSPROBLEM o problema de encontrar tal procedimento para determinar se uma fórmula do cálculo de predicados de primeira ordem é válida ou não.

Essa crença na tratabilidade de todo problema mate-

mático apareceu em seus trabalhos "Mathematical Problems" de 1900 e "On the infinite", de 1925 e perdurou até 1931 quando Gödel publicou seu famoso teorema da incompletude, que estabelecia que para todo sistema formal (axiomas e regras de inferência) consistente (isto é, no qual não é possível derivar A e $\neg A$, a partir dos axiomas do sistema, para qualquer fórmula A), existem proposições que são verdadeiras mas que não são prováveis dentro do sistema, ou seja, não são deriváveis dos axiomas usando as regras de inferência.

Na prova de seu teorema, Gödel usou uma classe de funções (recursivas primitivas) que mapeavam k -uplas de números naturais (números de Gödel) em números naturais. Gödel, em 1934, afirmou que essas funções tinham uma propriedade importante: para qualquer conjunto de valores para seus argumentos, a função podia ser calculada por um procedimento finito. Além disso estabeleceu também que o contrário parecia valer se fosse considerada uma classe maior de funções, definidas por Herbrand e Gödel (atualmente conhecidas como funções recursivas). Gödel observou entretanto que isto não podia ser provado porque a noção de computação efetiva não era definida.

Em 1936, Church empregou duas caracterizações de computabilidade efetiva (o cálculo- λ de Church e Kleene e as funções recursivas de Herbrand e Gödel) para provar que o entscheidungsproblem para a aritmética não era decidível. Independentemente de Church, também em 1936, Turing propôs um formalismo para a representação de procedimentos efetivos. A ideia de Turing foi muito importante porque era a primeira vez que se identificava a noção intuitiva de efetividade com programas escritos para uma máquina de calcular automática.

(hoje conhecida como máquina de Turing). Estes três formalismos, assim como outros modelos propostos posteriormente, foram todos mostrados serem equivalentes, levando a acreditar que qualquer um deles podia ser usado para caracterizar precisamente a noção de procedimento efetivo e função efetivamente calculável.

A teoria de linguagens formais foi desenvolvida inicialmente por Post. Em seu trabalho de 1943, Post mostrou que os sistemas formais podiam ser obtidos por transformações em seqüências de símbolos, a partir de um axioma (um símbolo) e de regras de produção da forma $xS \rightarrow Sy$ (com x, y símbolos e S uma seqüência qualquer de símbolos). Os sistemas de produção de Post foram fundamentais para o estudo de gramáticas formais.

Outro trabalho fundamental para o estudo de gramáticas foi o de Chomsky, que em 1959, motivado pelo problema de desenvolver uma gramática formal para o inglês e outras linguagens naturais, investigou algumas ideias novas. Embora Chomsky não tenha obtido sucesso, conseguiu estabelecer vários tipos de gramática formal assim como o relacionamento entre as classes de linguagens que os vários tipos de gramática representam. Este relacionamento é conhecido atualmente como HIERARQUIA DE CHOMSKY e é objeto dos demais capítulos.

1.2 - CONCEITOS BÁSICOS

(a) ALFABETOS E LINGUAGENS

definição 1.1 - um alfabeto (ou vocabulário) Σ é um conjunto finito, não vazio de símbolos.

definição 1.2 - uma palavra (ou cadeia) sobre um alfabeto Σ é uma sequência finita de símbolos de Σ .

x é uma cadeia $\Rightarrow x = a_1 a_2 \dots a_n$, $n \geq 0$

$$a_i \in \Sigma, i = 1, 2, \dots, n$$

quando $n=0$, $x = \epsilon$ (cadeia vazia)

definição 1.3 - o comprimento de uma cadeia x sobre um alfabeto Σ (representado por $|x|$) é o número de ocorrências de símbolos de Σ em x .

Por exemplo: $\Sigma = \{0, 1\}$

$$|011| = 3 \quad ; \quad |\epsilon| = 0$$

definição 1.4 - sejam x e y duas cadeias sobre Σ . A concatenação de x e y é definida como a cadeia xy .

Por exemplo: $\Sigma = \{0, 1\}$ $x = 010$ $y = 10$

$$xy = 01010$$

$$yx = 10010$$

definição 1.5 - Sejam X e Y , conjuntos de cadeias sobre Σ . O produto de X e Y é definido por:

$$XY = \{xy \mid x \in X, y \in Y\}$$

Notação:

$$X^0 = \{ \lambda \}$$

$$X^{i+1} = X^i X \quad (i \geq 0)$$

O fecho transitivo reflexivo (X^*) e o fecho transitivo (X^+) de um conjunto de cadeias X é então:

$$X^* = \bigcup_{i \geq 0} X^i$$

$$X^+ = \bigcup_{i \geq 1} X^i = X^* X$$

definição 1.6 - uma linguagem L é um conjunto qualquer de cadeias sobre um alfabeto Σ , ou seja $L \subseteq \Sigma^*$.

Como representar uma linguagem L ?

Se L é finito, uma maneira óbvia (embora possa não ser prática) é listar todas as suas cadeias. Mas se L for infinito?

Existem dois sistemas principais para representação de linguagens:

- (a) o sistema gerador (GRAMÁTICAS)
- (b) o sistema reconhecedor (AUTÔMATOS)

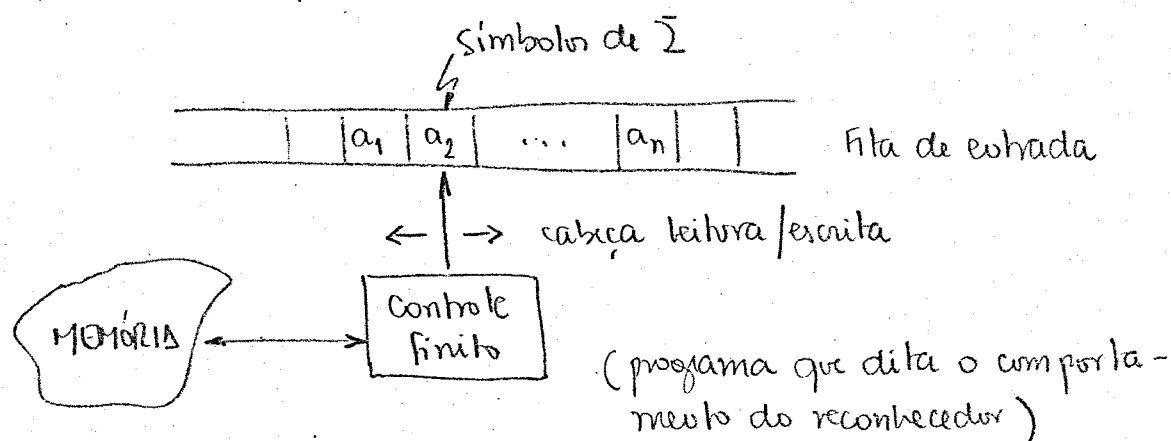
Neste trabalho estudam-se esses dois sistemas para cada uma das

classes de linguagens da hierarquia de Chomsky.

(b) RECONHECEDORES E GRAMÁTICAS

Um reconhecedor de uma linguagem L é a representação de um procedimento que quando apresentado a uma cadeia qualquer, pára e responde SIM após um número finito de passos, caso a cadeia pertença à linguagem L e pára e responde NÃO ou não pára, caso a cadeia não pertença a L .

Simbolicamente:



Uma configuração do reconhecedor é uma descrição:

- (a) do estado do controle finito
- (b) do conteúdo da fita de entrada e da posição da cabeça
- (c) do conteúdo da memória.

Um reconhecedor ACETA (ou RECONHECE) uma cadeia w se:

1. partindo de uma CONFIGURAÇÃO INICIAL (isto é, controle finito num estado inicial, fita de entrada contendo w , cabeça posi-

cionada no símbolo mais a esquerda de w , memória com conteúdo inicial)

2. faz uma sequência finita de "movimentos" e
3. termina numa configuração FINAL (controle finito num estado final e tendo sido lidos todos os símbolos de w)

A linguagem aceita (definida, ou reconhecida) por um reconhecedor R é:

$$L(R) = \{w \in \Sigma^* \mid R \text{ aceita } w\}$$

definição 1.7 - uma gramática é uma 4-upla $G = (N, \Sigma, P, S)$ onde:
 N é um conjunto finito não vazio (símbolos não terminais)

Σ é um conjunto finito não vazio (símbolos terminais)
 tal que $\Sigma \cap N = \emptyset$

$S \in N$ (símbolo inicial)

P é um conjunto de regras da forma $\alpha \rightarrow \beta$ onde

$$\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^* ; \beta \in (N \cup \Sigma)^*$$

(conjunto de produções)

Por exemplo:

$$G = (N, \Sigma, P, S) \text{ onde } N = \{A, S\}, \Sigma = \{a, b\}$$

$$\text{e } P = \{ S \rightarrow ab, S \rightarrow aASb, S \rightarrow bSb, AS \rightarrow bSb, A \rightarrow \perp, aASAb \rightarrow aa \}$$

definição 1.8 - Seja uma gramática $G = (N, \Sigma, P, S)$ e $V = N \cup \Sigma$.
Sejam $\alpha', \beta' \in V^*$.
 α' deriva diretamente β' ($\alpha' \Rightarrow \beta'$) se existem
 $\alpha_1, \alpha_2, \alpha, \beta \in V^*$ tais que

$$\alpha' = \alpha_1 \alpha \alpha_2 \quad ; \quad \beta' = \alpha_1 \beta \alpha_2 \quad \text{e} \quad \alpha \rightarrow \beta \in P.$$

Por exemplo, seja G do exemplo anterior:

$$S \Rightarrow ab$$

$$S \Rightarrow aASb \Rightarrow abSbb \Rightarrow ababbb \quad (S \stackrel{3}{\Rightarrow} ababbb)$$

Notação:

$\stackrel{n}{\Rightarrow}$ - deriva em n passos

$\stackrel{*}{\Rightarrow}$ - deriva em zero ou mais passos

definição 1.9 - Seja $G = (N, \Sigma, P, S)$ e $V = N \cup \Sigma$. O conjunto de formas sentenciais de G é definido por:

$$S(G) = \{ \alpha \in V^* \mid S \stackrel{*}{\Rightarrow} \alpha \}$$

definição 1.10 - Seja $G = (N, \Sigma, P, S)$ e $V = N \cup \Sigma$. A linguagem gerada (ou representada) por G é o conjunto

$$L(G) = \{ w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w \} = \Sigma^* \cap S(G)$$

Exemplo: gramática que gera o conjunto $L = \{ 0^n 1^n \mid n \geq 1 \}$

$$G = (N, \Sigma, P, S) \text{ onde } N = \{S\}, \Sigma = \{0, 1\}$$

$$\text{e } P = \{ S \rightarrow 01, S \rightarrow 0S1 \}$$

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

Exercício: provar que $L = L(G)$!

(c) HIERARQUIA DE CHOMSKY

A gramática como foi definida anteriormente (definição 1.7) é chamada gramática irrestrita ou gramática tipo-0.

definição 1.11 - $G = (N, \Sigma, P, S)$ com $V = N \cup \Sigma$ é gramática tipo-1 ou sensível ao contexto se toda produção de P é da forma:

$$i) \alpha A \gamma \rightarrow \alpha \beta \gamma \quad (A \in N; \alpha, \gamma \in V^*; \beta \in V^+)$$

ou

$$ii) S \rightarrow \perp \text{ e se essa produção ocorre, então } S \text{ não aparece no lado direito de qualquer produção.}$$

definição 1.12 - $G = (N, \Sigma, P, S)$ é gramática tipo-2 ou livre de contexto se toda produção de P é da forma:

$$A \rightarrow \alpha \quad (A \in N; \alpha \in V^*)$$

OBSERVAÇÃO: uma forma alternativa de gramática livre de contexto, usada na especificação de linguagens de programação é a forma normal de Backus, BNF (ou forma de Backus - Naur)

Em BNF:

- < > delimita símbolos não terminais
- ::= substitui \rightarrow
- | para definições alternativas ("ou")

Exemplo: gramática que representa o conjunto de inteiros

$I \rightarrow D$

$I \rightarrow ID$

$D \rightarrow 0$

$D \rightarrow 1$

\vdots

$D \rightarrow 9$

Em BNF:

$\langle \text{inteiro} \rangle ::= \langle \text{dígito} \rangle \mid \langle \text{inteiro} \rangle \langle \text{dígito} \rangle$

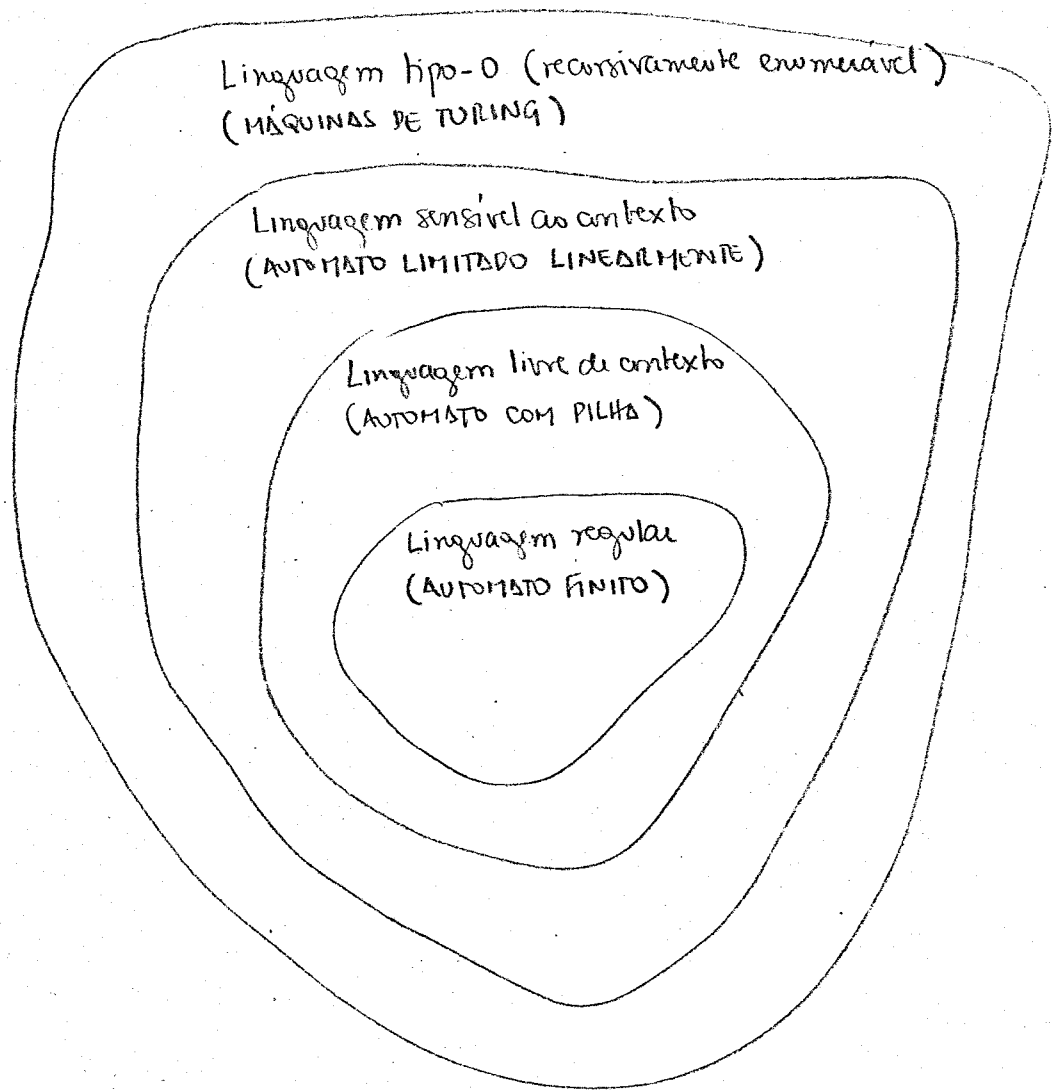
$\langle \text{dígito} \rangle ::= 0 \mid 1 \mid \dots \mid 9$

definição 1.13 - $G = (N, \Sigma, P, S)$ é uma gramática tipo-3 ou regular se toda produção de P é da forma:

$A \rightarrow aB$ ou $A \rightarrow a$ ou $S \rightarrow \epsilon$

$(A, B \in N ; a \in \Sigma)$

definição 1.14 - uma linguagem L é tipo- x se existe uma gramática tipo- x , G , tal que $L = L(G)$.



Nos próximos capítulos mostra-se que o relacionamento entre as várias classes de linguagens e os reconhecedores para cada uma delas é como mostrado na figura acima (hierarquia de Chomsky).