

As regras de inferência de um *SPP* podem ser bastante complexas. As operações básicas são de concatenação, casamento de padrões e substituição. Tais operações não fazem parte do sistema formal e são de difícil execução, mesmo quando o agente é um ser humano bem treinado. O leitor familiarizado com os processos de análise sintática para a construção de compiladores pode apreciar tais dificuldades. A complexidade da linguagem \mathcal{L} é um fator preponderante tanto para o uso do casamento de padrões como para a construção das regras de inferência. O leitor interessado pode consultar Book e Otto [81] que trata dos chamados sistemas de reescrita.

5.3 Linguagens Formais e Gramáticas

O aprendizado de uma língua qualquer envolve o estudo da estrutura das sentenças. A grosso modo esta estrutura é chamada de gramática para a língua, e sua apresentação usual é como um conjunto de critérios chamados de regras gramaticais que definem a correteza das sentenças. As regras gramáticas são sistemas formais geradores de linguagens, portanto deve-se esperar que uma gramática construa a língua.

Em geral, uma linguagem natural tende a ser muito extensa e complexa. Duas das atividades dos linguistas são definir com precisão as sentenças válidas de linguagens e encontrar formas estruturadas de representá-las. Entretanto, há uma dificuldade para definir completamente estas linguagens. Considere a seguinte sentença do português: *O menino louco pintava o lindo quadro*.

Pode-se dizer que a sentença é constituída de sujeito “O menino louco- e predicado - “pintava o lindo quadro”. Estes elementos podem ser mais analisados. O predicado é constituído de verbo “pintava” e objeto “lindo quadro”. A análise desta sentença é apresentada na Figura 5.2 e descreve uma representação conhecida como árvore de derivação sintática. Quando uma sentença produz uma árvore de derivação válida, diz-se que esta sentença é válida. Contudo, para uma árvore de derivação ser válida é preciso que ela seja especificada, uma tarefa pouco trivial quando se trata de uma linguagem natural. Uma tentativa incompleta de formalização das regras

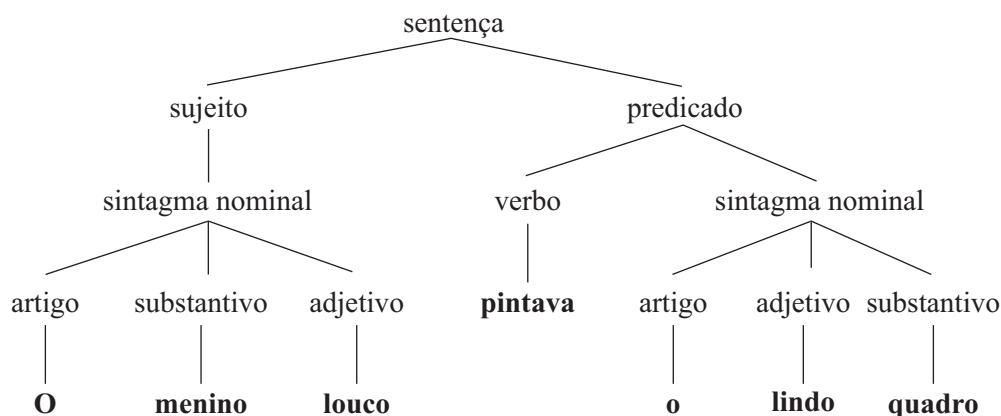


Figura 5.2: Árvore de Derivação Sintática de uma sentença em Português.

do Português pode ser descrita como se segue:

$\langle \textit{sentenca} \rangle \rightarrow \langle \textit{sujeito} \rangle \langle \textit{predicado} \rangle$
 $\langle \textit{sujeito} \rangle \rightarrow \langle \textit{sintagma nominal} \rangle$
 $\langle \textit{predicado} \rangle \rightarrow \langle \textit{verbo} \rangle \langle \textit{sintagma nominal} \rangle$
 $\langle \textit{sintagma nominal} \rangle \rightarrow \langle \textit{artigo} \rangle \langle \textit{substantivo} \rangle \langle \textit{adjetivo} \rangle$
 $\langle \textit{sintagma nominal} \rangle \rightarrow \langle \textit{artigo} \rangle \langle \textit{adjetivo} \rangle \langle \textit{substantivo} \rangle$
 $\langle \textit{verbo} \rangle \rightarrow \textit{pintava}$
 $\langle \textit{artigo} \rangle \rightarrow \textit{o}$
 $\langle \textit{substantivo} \rangle \rightarrow \textit{menino}$
 $\langle \textit{substantivo} \rangle \rightarrow \textit{quadro}$
 $\langle \textit{adjetivo} \rangle \rightarrow \textit{louco}$
 $\langle \textit{adjetivo} \rangle \rightarrow \textit{lindo}$

Estas regras indicam que os elementos à esquerda da seta podem ser transformados nos elementos à direita da seta. Os símbolos $\langle s \rangle$ indicam categorias gramaticais genéricas, chamados *símbolos não-terminais*, e os demais símbolos são palavras do Português, chamados *símbolos terminais*.

As regras podem estabelecer que a sentença dada é gramaticalmente correta, bastando constatar que esta é gerável a partir daquelas. Além disso, estas regras podem ser utilizadas para construir outras sentenças corretas do Português como “O menino lindo pintava o quadro louco”. Além disso, também podem ser produzidas

outras sentenças sem valor semântico como “O quadro louco pintava o lindo menino”.

Até o presente momento, não existe um conjunto de regras gramaticais e de palavras que possibilitem a formalização de uma linguagem natural, dada a explosão combinatorial que palavras e regras de uma linguagem natural produz. Entretanto, o mesmo conceito funciona para linguagens de programação pois estas podem ser definidas por uma sintaxe rígida e uma semântica bem determinada, possibilitando o processamento computacional.

Para especificar a sintaxe de uma linguagem de programação de modo sistemático é preciso uma gramática que possibilite:

- *Geração*: um método sistemático de especificação para a construção de programas corretos;
- *Reconhecimento*: um método de análise de um programa a fim de verificar se ele está sintaticamente correto.

A definição seguinte torna precisa a noção de gramática como um sistema formal.

Definição 5.5 Uma gramática \mathcal{G} (Gramática de Chomsky³) é um sistema formal $\langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$, onde:

Σ É um alfabeto consistindo de dois subconjuntos disjuntos N e T , chamados de alfabeto não terminal e terminal, respectivamente.

\mathcal{L} É o conjunto Σ^*

\mathcal{A} É o conjunto unitário $\{S\}$, sendo $S \in N$. S é dito ser o símbolo de partida e possui o nome da categorial sintática principal.

\mathcal{R} É um conjunto de relações binárias em \mathcal{L} , chamadas de regras de produção.

³Avram Noam Chomsky (1928-) é responsável pela gramática gerativa transformacional, abordagem que revolucionou os estudos no domínio da linguística teórica. É também o autor de trabalhos fundamentais sobre as propriedades matemáticas das linguagens formais, sendo o seu nome associado à chamada Hierarquia de Chomsky.

Por convenção, os símbolos não terminais são representados por letras maiúsculas e os símbolos terminais, por letras minúsculas. Seja a gramática $G1$ com $N = \{S, A\}$, $T = \{a, b\}$, $\mathcal{A} = \{S\}$ e $\mathcal{R} = \{S \rightarrow aA, A \rightarrow b\}$. Neste caso, $S \rightarrow aA \rightarrow ab$ que é a única cadeia gerada por $G1$.

Definição 5.6 Seja $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ uma gramática, $x, y \in \mathcal{L}$. Diz-se que:

(a) y é diretamente derivável em \mathcal{G} a partir de x , o que é denotado por $x \Rightarrow_{\mathcal{G}} y$ se e somente se

$$x = x_1 \alpha x_2 \text{ e } y = x_1 \beta x_2 \text{ e } \alpha \rightarrow \beta \in \mathcal{R}$$

(b) y é derivável em \mathcal{G} a partir de x , o que é denotado por $x \Rightarrow_{\mathcal{G}}^* y$ se e somente se $y = x$ ou existem $\beta_1, \beta_2, \dots, \beta_l$ tais que $\beta_l = \beta$ e para todo i , $1 \leq i < l$ $\beta_i \Rightarrow_{\mathcal{G}} \beta_{i+1}$

Uma gramática pode ser considerada como um SPP , cujas regras de produção da forma $\alpha \rightarrow \beta$ são representadas como $\boxed{1}\alpha\boxed{2} \rightarrow \boxed{1}\beta\boxed{2}$. Seja $G2$ com $N = \{A, B\}$, $T = \{a, b, c\}$, $\mathcal{A} = \{A\}$ e $\mathcal{R} = \{A \rightarrow aB, B \rightarrow bB, B \rightarrow c\}$. A sentença $x = abbbc$ é uma produção de $G2$ porque ela é derivável da seguinte forma: $A \xRightarrow{1} aB \xRightarrow{2} abB \xRightarrow{2} abbB \xRightarrow{2} abbbB \xRightarrow{3} abbbc$.

Definição 5.7 Seja $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ uma gramática. A linguagem formal gerada por \mathcal{G} , denotada por $L(\mathcal{G})$ é o conjunto de teoremas de \mathcal{G} contendo apenas símbolos terminais de \mathcal{G} . Assim $L(\mathcal{G}) = \{x \in T^* | S \Rightarrow_{\mathcal{G}}^* x\}$.

Exemplo 5.5 Seja a gramática $G2$ definida anteriormente. Pode-se verificar que $L(G2) = \{x \in T^* | x = ab^n c, n \geq 0\}$.

Exemplo 5.6 Seja a gramática $G3$ com $N = \{S\}$, $T = \{a, b, c\}$, $\mathcal{A} = \{S\}$ e $\mathcal{R} = \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow c\}$. Então é fácil verificar que:

$$L(\mathcal{G}) = \{c, aca, aacaa, bcb, bacab, bbcbb, \dots\}$$

Exemplo 5.7 Seja a gramática $G4$ com $N = \{E, T, F\}$, $T = \{+, *, (,), x\}$, $\mathcal{A} = \{E\}$ e $\mathcal{R} = \{E \rightarrow E+T, E \rightarrow T, T \rightarrow T*F, T \rightarrow F, F \rightarrow (E), F \rightarrow x\}$. Um exemplo de cadeia derivável da gramática é a expressão conforme descrito na derivação $E \xRightarrow{2}$

$$T \xrightarrow{3} T + F \xrightarrow{5} T * (E) \xrightarrow{5} F * (E) \xrightarrow{1} F * (E + T) \xrightarrow{2} F * (T + T) \xrightarrow{4} F * (F + F) \xrightarrow{6} x * (x + x).$$

Quando se trata de linguagem de programação, um impedimento importante é a impossibilidade de uma cadeia $w \in L(G)$ possuir mais de um entendimento. Assim, linguagens de programação não devem ser ambíguas, ou ao menos, devem permitir que eventuais ambiguidades possam ser facilmente evitadas. O mais notório caso de ambiguidade é conhecido como o *else pendente* (dangling else).

Seja um gramática \mathcal{G} com o seguinte conjunto de regras:

$$BLK \rightarrow \text{if } a \text{ then } BLK \text{ else } BLK$$

$$BLK \rightarrow \text{if } a \text{ then } BLK$$

$$BLK \rightarrow b$$

A gramática \mathcal{G} é ambígua uma vez que a cadeia “if a then if a then b else b” pode ser interpretada como “if a then (if a then b else b)” ou como “if a then (if a then b) else b”. Alguns compiladores não aceitam este tipo de construção, e outros, quando aceitam, escolhem a interpretação que associa o *else* ao *if* mais próximo, isto é, “if a then (if a then b else b)”.

Em alguns casos, é possível perceber esta ambiguidade como algo inerente à gramática e nestas situações, pode-se reescrever a mesma a fim de eliminar estas ambiguidades. Seja a gramática $G5$ com $N = \{E, T, F, A\}$, $T = \{+, *, 0, 1, 2, \dots, 9\}$, $\mathcal{A} = \{E\}$ e \mathcal{R} dado por:

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow 0 \dots 9$$

que é utilizada para gerar expressões aritméticas com $+$ e $*$ para números inteiros. Este tipo de construção é bastante empregada em linguagens de programação. Além disto, seja também a gramática $G6$ com $N = \{E, A\}$, $T = \{+, *, 0, 1, 2, \dots, 9\}$, $\mathcal{A} = \{E\}$ e \mathcal{R} dado por:

$$E \rightarrow E + E$$

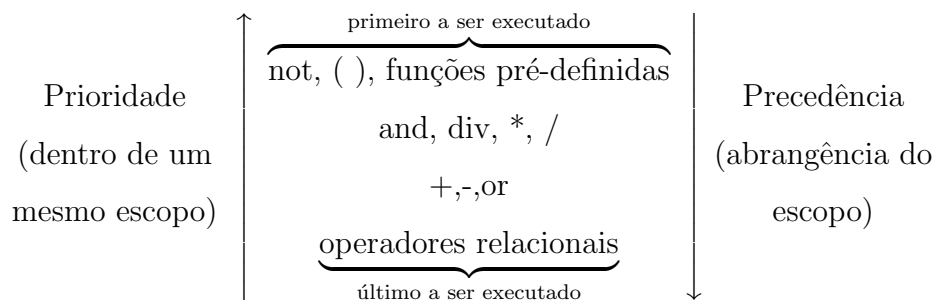
$$E \rightarrow E * E$$

$$E \rightarrow F$$

$$F \rightarrow 0 \dots 9$$

que não só é equivalente a $G5$, como também possui uma quantidade menor de produções e símbolos. Entretanto, $G6$ é ambígua enquanto que $G5$, não. Por exemplo, a cadeia $2 + 4 * 6$ possui dois entendimentos na gramática $G6$, “ $2+(4*6)$ ” e “ $(2+4)*6$ ”. Já na gramática $G5$ só há o entendimento “ $2+(4*6)$ ”.

Para dirimir a ambiguidade é preciso compreender as propriedades de precedência e de associatividade. A precedência permite decidir qual o correto entendimento das expressões, isto é, quanto maior a precedência de um operador, maior será o escopo contemplado por este mesmo operador. Já a associatividade permite decidir o correto entendimento de operações contendo operadores de mesma precedência. Em geral, a relação de prioridade e precedência entre operadores segue o esquema a seguir.



Por exemplo, na gramática $G5$ o operador $+$ tem maior precedência que $*$ porque está definido mais próximo das regras de produção que se iniciam por axiomas. Além disso, $*$ tem maior prioridade que $+$ pois é resolvido primeiro que $+$. O exemplo a seguir demonstra a precedência de operadores da gramática $G5$.

Exemplo 5.8 Precedência de operadores da gramática $G5$:

$$2 + 4 * 6 = +(2, *(4, 6))$$

$$2 * 4 + 6 * 8 = +(* (2, 4), * (6, 8))$$

$$2 + 4 + 6 = +(+ (2, 4), 6)$$

Assim, o truque é aumentar a precedência dos operadores, colocando suas regras cada vez mais distantes do axioma \mathcal{A} . No exemplo a seguir, a primeira

gramática é inicialmente ambígua. Na segunda gramática, a precedência é resolvida pela associatividade que é colocada em um nível mais baixo da árvore sintática. Contudo, este procedimento delega a quem monta a expressão a responsabilidade de determinar a precedência entre as operações utilizando a associatividade pois os operadores de $+$ e $*$ têm a mesma precedência. Por fim, a última gramática resolve este problema colocando o operador $*$ com maior precedência, isto é, colocando o operador $*$ em um nível mais baixo da árvore sintática que o operador $+$.

1ª GRAMÁTICA	2ª GRAMÁTICA	3ª GRAMÁTICA
	Ambiguidade removida com associatividade, $+$ e $*$ tem a mesma	
Gramática ambígua	precedência	Precedência convencional
$E \rightarrow E + E$	$E \rightarrow E + T$	$E \rightarrow E + T$
$E \rightarrow E * E$	$E \rightarrow E * T$	$E \rightarrow T$
$E \rightarrow (E)$	$E \rightarrow T$	$T = T * F$
$E \rightarrow a$	$T \rightarrow (E)$	$T \rightarrow F$
	$T \rightarrow a$	$F \rightarrow (E)$
		$F \rightarrow a$

5.4 Hierarquia de Chomsky

Formas normais impõem restrições às formações de regras de uma gramática a fim de assegurar que determinadas propriedades sejam alcançadas. Existem alguns tipos de abordagens para essa construção das restrições, sendo duas de maior destaque: a forma normal de Chomsky e a de Greibach. Nesta seção será usada a abordagem de Chomsky. Sob esta ótica, uma gramática $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ pode ser classificada pela forma de suas regras de produção em quatro tipos:

Tipo 0 Nenhuma restrição é imposta na forma das regras de produção. Estas gramáticas são chamadas de Turing reconhecíveis. A definição 5.5 refere-se a este tipo.

Tipo 1 Se $\alpha \rightarrow \beta \in \mathcal{R}$ tem-se que $|\alpha| \leq |\beta|$, onde $|\alpha|$ e $|\beta|$ representam os com-

primitos de α e β , respectivamente, e excetuando a regra $\alpha \rightarrow \Lambda$, cujo α não pode estar do lado direito de qualquer produção. Gramáticas do tipo 1 são também chamadas *sensíveis ao contexto* pois há restrição de que as regras sejam da forma $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ com α_1, α_2 e $\beta \in \Sigma^*$, $\beta \neq \Lambda$ e $A \in N$. Esta restrição é equivalente à anterior, e pode ser lida como “ A pode ser substituído por β no contexto de α_1 e α_2 ”. As gramáticas dos exemplos a seguir são do tipo 1.

Exemplo 5.9 Seja a gramática \mathcal{G} com $N = \{S\}$, $T = \{0, 1\}$ e $\mathcal{R} = \{S \rightarrow 0S1, S \rightarrow 01\}$. Então é fácil verificar que $L(\mathcal{G}) = \{0^n 1^n | n \geq 1\}$.

Exemplo 5.10 Seja a gramática \mathcal{G} com $N = \{S, B, C\}$ e $T = \{a, b, c\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. S \rightarrow aSBC & 5. bB \rightarrow bb \\ 2. S \rightarrow aBC & 6. bC \rightarrow bc \\ 3. CB \rightarrow BC & 7. cC \rightarrow cc \\ 4. aB \rightarrow ab \end{array} \right\}$$

Neste caso não é tão imediato verificar que $L(\mathcal{G}) = \{a^n b^n c^n | n \geq 1\}$. Deixe-se como exercício para o leitor (um arrazoado completo para este exemplo encontra-se em Hopcroft e Velman [82], seção 2.2).

Tipo 2 Se $\alpha \rightarrow \beta \in \mathcal{R}$ tem-se que $\alpha \in N$. Em contraste com as gramáticas do tipo 1, estas são chamadas “livres do contexto”. A gramática do exemplo a seguir é do tipo 2.

Exemplo 5.11 Seja a gramática \mathcal{G} com $N = \{S, A, B\}$ e $T = \{a, b\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. S \rightarrow aB & 5. A \rightarrow bAA \\ 2. S \rightarrow bA & 6. B \rightarrow b \\ 3. A \rightarrow a & 7. B \rightarrow bS \\ 4. A \rightarrow aS & 8. B \rightarrow aBB \end{array} \right\}$$

A linguagem $L(\mathcal{G})$ consiste das cadeias em $T^* - \{\Lambda\}$ com mesmo número de símbolos a e b .

A gramática do exemplo 5.9 também é do tipo 2.

Tipo 3 Também chamadas de gramáticas regulares, nestas gramáticas toda regra de produção é de uma das formas a seguir:

$$A \rightarrow a, \quad a \in T \text{ e } A \in N$$

$$A \rightarrow aB, \quad a \in T \text{ e } A, B \in N$$

Exemplo 5.12 Seja a gramática \mathcal{G} com $N = \{S, A, B\}$ e $T = \{0, 1\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. \ S \rightarrow 0A & 6. \ B \rightarrow 1B \\ 2. \ S \rightarrow 1B & 7. \ B \rightarrow 1 \\ 3. \ A \rightarrow 0A & 8. \ B \rightarrow 0 \\ 4. \ A \rightarrow 0S & 9. \ S \rightarrow 0 \\ 5. \ A \rightarrow 1B & \end{array} \right\}$$

Naturalmente as gramáticas do tipo 3 são do tipo 2, 1 e 0; as do tipo 2 são também 1 e 0 e as do tipo 1 são também 0.

Existe uma *caracterização* sintática do poder gerativo e representacional com respeito ao *tipo* de linguagem gerada, e uma correspondência efetiva entre tais gramáticas e certos sistemas formais conhecidos como *reconhecedores*.

Os reconhecedores são sistemas de *validação* que não produzem *saídas*. Em geral, admitem certos *estados iniciais* e quando *param* é verificado em que *estado final* se encontram. Em geral possuem apenas dois estados finais *aceitação* e *rejeição*. No entanto existem sistemas que possuem um número ilimitado de estados finais.

As gramáticas de tipo 3 são as menos complexas e as linguagens que elas geram são chamadas linguagens regulares: seus elementos podem ser reconhecidos em tempo real por autômatos finitos. O estudo dos autômatos finitos não será feito neste capítulo, mas eles surgem naturalmente a partir das regras de produção das gramáticas tipo 3.

Exemplo 5.13 Seja a gramática tipo 3 com

$$\Sigma = \{a, b, S, A, B\}$$

$$N = \{S, A, B\}$$

$$\mathcal{A} = \{S\}$$

$$\mathcal{R} = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow bB, B \rightarrow aA, A \rightarrow a\}$$

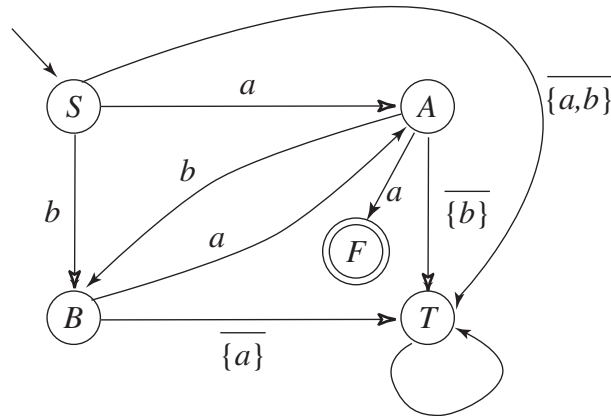


Figura 5.3: Autômato finito.

tem como reconhecedor o autômato finito da Figura 5.3

O estado S é o estado inicial e o estado F é o estado final, dada uma palavra, por exemplo $ababa$. Esta é lida da esquerda para a direita, símbolo a símbolo. Quando lê a a máquina vai para o estado A , quando lê b vai para o estado B , lê a vai para o estado A , lê b vai para B e finalmente lê a e termina no estado A . T é uma armadilha.

A organização das linguagens através desta caracterização sintática é a chamada **Hierarquia de Chomsky**, que é apresentada esquematicamente pela Figura 5.4.

5.5 Gramáticas Sensíveis ao Contexto

Quando Chomsky introduziu o conceito de gramáticas sensíveis ao contexto, ele queria capturar a idéia de que em linguagens naturais, certas palavras podem ser, ou não ser, apropriadas em determinadas posições, de acordo com o contexto onde elas se encontram, isto é, os strings vizinhos a estas palavras. Assim a substituição de um símbolo terminal depende de um string de contexto do seu lado esquerdo e de outro string de contexto do seu lado direito.

Estas mesmas gramáticas sensíveis ao contexto são suficientes para descrever

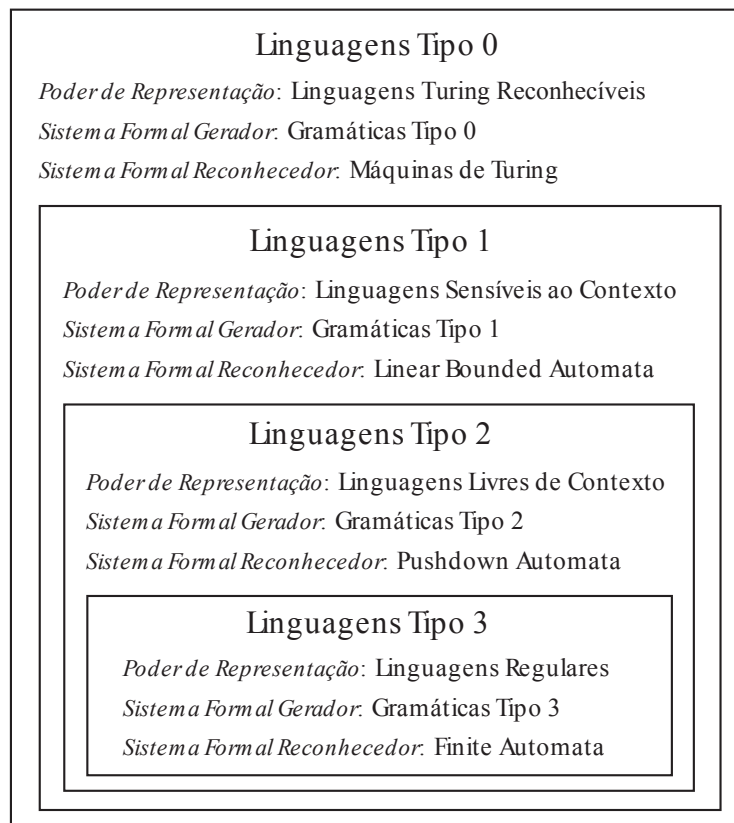


Figura 5.4: Hierarquia de Chomsky.

Referências Bibliográficas

- [1] HOPCROFT, J. E., *Theory os Machines and Compuatations*, chapter An n log n algorithm for minimizing states in a finite automaton, Academic Press, pp. 189–196, 1971.
- [2] MONTEIRO, A. A., PAULO, J. D. S., *Aritmética Racional*. Lisboa, Livraria Avelar Machado, 1945.
- [3] IFRAH, G., *Os Números: a história de uma grande invenção*. São Paulo, Globo S.A., 2005.
- [4] DEDEKIND, R., *Was sind und was sollen die Zahlen*, v. 3, *Gesammelte Mathematische Werke*. New York, Chelsea Publishing Company, 1969. pp. 335-391.
- [5] GÖDEL, K., *Collected Works*, v. 2, *Gesammelte Mathematische Werke*. Oxford, Oxford University Press, 1990.
- [6] ISRAEL, D., “Reflections on Gödel’s and Gandy’s Reflection on Turing’s Thesis”, *Minds and Machines*, v. 12, n. 2, pp. 181–201, 2002.
- [7] SOBRINHO, J. Z., “Aspectos da Tese de Church-Turing”, *Revista Matemática Universitária - USP*, v. 1, n. 6, pp. 1–23, 1987.
- [8] MCDERMOTT, D., “Artificial Intelligence Meets Natural Stupidity”, *SI-GART Newsletter*, v. 57, pp. 4–9, April 1976.
- [9] SETTI, M. D. O. G., *O Processo de Discretiza çã o do Raciocínio Matemático na Tradu çã o para o Raciocínio Computacional*, Report, Universidade Federal do Paraná, 2009.

- [10] GUERREIRO, G., “A Vanguarda Matemática e os Limites da Razão”, *Scientific American Brasil*, v. 5, n. 12, pp. 39–56, 2007. Coleção Gênios da Ciência.
- [11] SMULLYAN, R., *Gödel’s Incompleteness Theorems*. Oxford University Press, 1992.
- [12] GÖDEL, K., *The Undecidable*, chapter On Formally Undecidable Propositions of Principia Mathematica and Related Systems, New York, Raven Press, pp. 5–38, 1965.
- [13] WHITEHEAD, A. N., RUSSELL, B., *Principia Mathematica*. Londres, Cambridge University Press, 1913.
- [14] NAGEL, E., NEWMAN, J. R., *Gödel’s Proof*. USA, Routledge, 1989.
- [15] GÖDEL, K., “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme - On Formally Undecidable Propositions of Principia Mathematica and Related Systems”, *Kurt Gödel: Collected Works*, v. 1, n. 1, pp. 144–195, 1986. Tradução para o inglês por Martin Hirzel. www.research.ibm.com/people/h/hirzel/papers/canon00-goedel.pdf [capturado em 13 de agosto de 2011].
- [16] MELO, A. C. V. D., SILVA, F. S. C. D., *Modelos Clássicos de Computação*, Coleção Schaum. São Paulo, Thomson, 2006.
- [17] KUBRUSLY, R. D. S., *Uma viagem informal ao Teorema de Gödel ou (O preço da matemática é o eterno matemático)*, Report, Universidade Federal do Rio de Janeiro, 2007. IM/UFRJ.
- [18] SMULLYAN, R., *Recursion Theory for Metamathematics*. Oxford University Press, 1993.
- [19] SMULLYAN, R., *What’s the Name of This Book*. Penguin Books, 1978.
- [20] SMULLYAN, R., *Forever Undecided: A Puzzle Guide to Gödel*. Oxford University Press, 1987.
- [21] GOLDSTEIN, R., *Incompleteness: The Proof and Paradox of Kurt Gödel*. W. W. Norton Company, Inc., 2005.

- [22] HOFSTADTER, D. R., *Gödel, Escher e Bach: an Eternal Golden Braid*. Nova Iorque, Basic Books, 1979.
- [23] Rodríguez-Consuegra, F. A. (ed.), *Kurt Gödel - Unpublished Philosophical Essays*. Berlin, Birkhäuser Verlag, 1995.
- [24] Feferman, S., *et al.* (eds.), *Kurt Gödel - Collected Works*, v. I, II, III. New York, Oxford University Press, 1986.
- [25] WANG, H., *Reflections on Kurt Gödel*. Cambridge, Massachusetts, The MIT Press, 1988.
- [26] SUPPES, P., *Axiomatic Set Theory*. New York, Dover, 1972.
- [27] LIPSCHUTZ, S., *Teoria Elementar dos Conjuntos*, Cole çã o Schaum. Sã o Paulo, McGraw-Hill do Brasil, 1990.
- [28] SUPPES, P., *Axiomatic Theory Set*. USA, Van Nostrand Company Inc., 1960.
- [29] MELLO, F. L. D., CARVALHO, R. L. D., “Knowledge Geometry”, *Journal of Information and Knowledge Management*, v. 14, pp. 1550028, 2015.
- [30] STOLL, R. R., *Set Theory and Logic*. USA, Dover Publications Inc., 1961.
- [31] MIRAGLIA, F., *Teoria dos Conjuntos: um mínimo*. Brasil, Edusp - Editora da Universidade de São Paulo, 1992.
- [32] HALMOS, P., *Teoria Ingênua dos Conjuntos*. Brasil, Edusp - Editora da Universidade de São Paulo, 1970.
- [33] GRATZER, G., *Universal Algebra*. USA, Van Nostrand Company Inc., 1968.
- [34] COHN, P. M., *Universal Algebra*. USA, Harper and Row, 1965.
- [35] GALLIER, J. H., *Logic for Computer Science*. USA, John Wiley and Sons Inc., 1987.
- [36] PREPARATTA, F. P., YEH, R. T., *Introduction to Discrete Structures for Computer Science and Engineering*. USA, Addison-Wesley Publishing Company, 1973.

- [37] SHOENFIELD, J. R., *Degrees of Unsolvability*. North-Holland Publishing Company, 1971.
- [38] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. USA, John Wiley and Sons, 1974.
- [39] EILENBERG, S., ELGOT, C., *Recursiveness*. New York: Academic Press, 1970.
- [40] CARVALHO, R. L. D., OLIVEIRA, C. M. G. M. D., *Modelos de Computação e Sistemas Formais*, 11^a Escola de Computação. Rio de Janeiro, Universidade Federal do Rio de Janeiro, 1998.
- [41] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. John Wiley & Sons, 1974.
- [42] KLEENE, S. C., *Introduction to Metamathematics*. D. Van Nostrand Company, Inc., 1952.
- [43] Rogers Jr., H., *Theory of Recursive Functions and Effective Computability*. USA, McGraw-Hill Book Company, 1967.
- [44] DAVIS, M., *Computability and Unsolvability*. New York, Dover, 1983.
- [45] BOOLOS, G. S., JEFFREY, R. C., *Computability and Logic*. Cambridge University Press, 1974.
- [46] MARTIN DAVIS, R. S., WEYUKER, E. J., *Computability Complexity and Languages*. New York, Academic Press, 1994.
- [47] MALLOZZI, J. S., LILLO, N. J. D., *Computability with Pascal*. New Jersey, Prentice-Hall, Inc., 1984.
- [48] TURING, A. M., *The Undecidable*, chapter On Computable Numbers, with an Application to the Entscheidungsproblem, New York, Raven Press, pp. 115–151, 1965.
- [49] ELGOT, C. C., ROBINSON, A., “Random-access stored-program machines, an approach to programming languages”, *Journal of the ACM*, v. 11, pp. 365–399, 1964.

- [50] PREPARATTA, F. P., YEH, R. T., *Introduction to Discrete Structures for Computer Science and Engineering*. Addison-Wesley Publishing Company, 1973.
- [51] HENNIE, F., *Introduction to Computability*. Addison-Wesley Publishing Company, 1977.
- [52] NELSON, R. J., *Introduction to Automata*. USA, Jonh Wiley & Sons, Inc., 1968.
- [53] CARVALHO, R. L. D., *Máquinas, Programas e Algoritmos*, 2^a Escola de Computação. Campinas, Universidade Estadual de Campinas, 1981.
- [54] MENDELSON, E., *Introduction to Mathematical Logic*, Cole Mathematics Series. 3 ed. The Wadsworth and Brooks, 1987.
- [55] MANIN, Y. I., *A Course in Mathematical Logic*, Graduate Texts in Mathematics 53. 1 ed. Springer-Verlag, 1977.
- [56] HOMER, S., SELMAN, A. L., *Computability and Complexity Theory*, Texts in Computer Science. 2 ed. Springer, 2011.
- [57] CUTLAND, N. J., *Computability: An introduction to recursive function theory*. Cambridge University Press, 1980.
- [58] SIPSER, M., *Introduction to The Theory of Computation*, Course Technology Series. 2 ed. Thomson, 2006.
- [59] WALTER CARNIELLI, R. L. E., *Computability: computable functions, logic and the foundations of mathematics*. Belmont, Wadsworth and Brooks, 1989.
- [60] TARSKI, A., *Logic, semantics, metamathematics*, chapter Fundamental concepts of the methodology of the deductive sciences, London, Oxford at the Clarendon Press, pp. 60–109, 1969.
- [61] ADAM YOUNG, M. Y., *Malicious Cryptography: Exposing Cryptovirology*. John Wiley and Sons Inc., 2004.

- [62] BONFANTE, G., KACZMAREK, M., MARION, J.-Y., *A Classification of Viruses through Recursion Theorems*, volume 4497 of Lecture Notes in Computer Science. 2 ed. CiE 2007, 2007.
- [63] MACHTEY, M., YOUNG, P., *An Introduction to the General Theory of Algorithms*. New York, North Holland, 1978.
- [64] ROBINSON, J. A., “A machine oriented logic based on the resolution principle”, *J. Assoc. Comput.*, v. 12, pp. 23–41, 1965.
- [65] ROBINSON, J. A., “Automatic deduction with hyper-resolution”, *Internat. J. Comput. Math.*, v. 1, pp. 227–234, 1965.
- [66] GILMORE, P. C., “A proof method for quantification theory: Its justification and realization”, *IBM Journal of Research and Development*, v. 4, n. 1, pp. 28–35, 1960.
- [67] DAVIS, M., PUTNAM, H., “A computing procedure for quantification theory”, *Journal of the ACM*, v. 7, n. 3, pp. 201–215, 1960.
- [68] CHANG, C.-L., LEE, R. C.-T., *Symbolic Logic and Mechanical Theorem Proving*. Academic Press Inc, 1973.
- [69] KLEENE, S. C., *Mathematical Logic*. Wiley, 1967.
- [70] HILBERT, D., ACKERMANN, W., *Prinmciples of Mathematical Logic*. Chelsea, 1950.
- [71] MCCAWLEY, J. D., *Everything That Linguists Have Always Wanted To Know About Logic*. 2 ed. The University of Chicago Press, 1993.
- [72] SUPPES, P., *Introduction to Logic*. D. van Nostrand, 1966.
- [73] RUSSELL, B., *A Filosofia do Atomismo Lógico*, *Lógica e Conhecimento*. 1 ed. Abril Cultural, 1974. (Os Pensadores, 42).
- [74] RUSSELL, B., *Significado e Verdade*. 1 ed. Zahar, 1978.
- [75] POPPER, K. R., *A Lógica da Pesquisa Científica*. 2 ed. Cultrix, 1974.

- [76] LAKATOS, I., , MUSGRAVE, A., *A Crítica e o Desenvolvimento do Conhecimento*. 1 ed. EDUSP, Cultrix, 1979. Tradução: M. O. Caiado.
- [77] WANG, H., *From Mathematics to Philosophy*. 1 ed. Cultrix, 1974.
- [78] GREEN, C. C., *The Application of Theorem Proving to Question-Answering Systems*. Ph.D. dissertation, Stanford, June 1969. AI Project MEMO AI-96.
- [79] LOVELAND, D. W., *Automated Theorem Proving: a Logical Basis*. 1 ed. North Holland, 1978.
- [80] HUGHES, G. E., LONDEY, D. G., *The Elements of Formal Logic*. USA, Methuen and Co Ltd, 1965.
- [81] BOOK, R. V., OTTO, F., *String-Rewriting Systems*. USA, Springer-Verlag, 1993.
- [82] HOPCROFT, J. E., ULLMAN, J. D., *Introduction to Automata Theory, Language and Computation*. USA, Addison-Wesley Publishing Company, 1979.
- [83] HOPCROFT, J. E., ULLMAN, J. D., *Formal Languages and their Relation to Automata*. USA, Addison-Wesley Publishing Company, 1969.
- [84] CURRY, H. B., *Foundations of Mathematical Logic*. New York, Academic Press, 1977.
- [85] SMULLYAN, R., *Theory of Formal Systems*. USA, Princeton, 1961.
- [86] BERSTEL, J., BOASSON, L., CARTON, O., *et al.*, *Handbook of Automata: from Mathematics to Applications*, chapter Minimization of automata, European Mathematical Society, pp. 189–196, 2010.
- [87] BEAL, M. P., CROCHEMORE, M., “Minimizing incomplete automata”, *Workshop on Finite State Methods and Natural Language Processing*, , september 2008. Ispra.
- [88] VALMARI, A., LEHTINEN, P., “Efficient minimization of DFAs with partial transition”, *Proc. 25th Symp. Theoretical Aspects of Comp. Sci.*, v. 08001, pp. 645–656, 2008. S. Albers and P. Weil, editors.

- [89] PAPADONIKOLAKIS, M., BOUGANIS, C.-S., CONSTANTINIDES, G.,
“Performance comparison of GPU and FPGA architectures for the SVM training problem”, *IEEE International Conference on FieldProgrammable Technology*, pp. 388–391, 2009.
- [90] MU, S., WANG, C., LIU, M., *et al.*, “Evaluating the potential of graphics processors for high performance embedded computing”, *Proc. IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 709–714, 2011.
- [91] KAI HWANG, F. A. B., *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.
- [92] AGARWAL, P., KRISHNAN, S., MUSTAFA, N., *et al.*, *Algorithms - ESA 2003, Lecture Notes in Computer Science*, chapter Streaming Geometric Optimization Using Graphics Hardware, Springer Berlin-Heidelberg, pp. 115–151, 2003.
- [93] TANENBAUM, A. S., *Organização Estruturada de Computadores*. 3 ed. Prentice Hall do Brasil, 1997.
- [94] BACKUS, J., “Can Programming be Liberated from von Neumann Style? A functional style and its algebra of program”, *ACM Turing Award Lecture, Communications of the ACM*, v. 21, n. 8, pp. 613–641, 1978.
- [95] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., *et al.*, “A Survey of General-Purpose Computing on Graphics Hardware”, *Eurographics 2005, State of the Art Reports*, pp. 21–51, 2005.
- [96] GUSTAFSON, J. L., “Reevaluating Amdahl’s law”, *Communications of the ACM*, v. 5, n. 31, pp. 532, 1988.
- [97] HANDLER, W., *Parallel Processing Systems, an advanced course*, chapter Innovative computer architecture - how to increase parallelism but not complexity, Cambridge University Press, pp. 1–41, 1982.
- [98] LOBUR, J., NULL, L., *The Essentials of Computer Organization And Architecture*. Jones and Bartlett Pub, 2006.

- [99] LEWIS, H. R., PAPADIMITRIOU, C. H., *Elements of the Theory of Computation*. 2 ed. New York, Prentice-Hall, 1998.
- [100] DUNNE, P., *Computability Theory: Concepts and Applications*. Ellis Horwood, 1991.
- [101] AARONSON, S., “NP-complete Problems and Physical Reality”, *ACM SIGACT News*, , march 2005. Complexity Theory Column 46.