



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SÃO PAULO

ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO - RECONHECEDOR DE GRAMÁTICAS

Linguagens Formais e Autômatos (GRULFAT)

Projeto: Projeto de Programação 1

Carolina Yumi Siroma - GU3042049

Celine Galdino Da Silva - GU3046354

Diogo Da Silva Almeida - GU3059995

Guarulhos/SP
2025

Carolina Yumi Siroma (GU3042049)

Celine Galdino Da Silva (GU3046354)

Diogo da Silva Almeida (GU3059995)

RELATÓRIO - RECONHECEDOR DE GRAMÁTICAS

Linguagens Formais e Autômatos (GRULFAT)

Matéria feita junto ao curso de
ENGENHARIA DE COMPUTAÇÃO do
Instituto Federal de Educação, Ciência e
Tecnologia de São Paulo - Campus Guarulhos.

Professor: Doutor Thiago Schumacher Barcelos
Professora: Doutora Alexandra Aparecida de Souza

Sumário

1	Introdução	1
2	Objetivos	1
3	Justificativa	1
4	Fundamentação Teórica	2
4.1	Linguagens Formais e Gramáticas	2
4.2	Componentes de uma Gramática Formal	3
4.3	Gramáticas Livres de Contexto (GLC)	3
4.4	Autômatos e Linguagens Formais	3
4.5	Validação de Gramáticas	4
5	Metodologia	4
5.1	Estrutura do Programa	4
5.2	Validação das Regras	4
6	Desenvolvimento e Implementação	5
6.1	Classificação de Símbolos	5
6.2	Validação das Regras	5
6.3	Identificação da Raiz	7
6.4	Processamento da Gramática	7
6.5	Liberação de Memória	8
7	Conclusão	8

1 Introdução

Este relatório descreve o desenvolvimento do projeto ”**Reconhecedor de Gramáticas**”, realizado como parte da disciplina *Linguagens Formais e Autômatos (GRULFAT)*. O objetivo deste projeto foi implementar um programa na linguagem C capaz de ler, interpretar e validar gramáticas formais, detectando erros nas regras de produção e identificando corretamente o símbolo inicial da gramática.

Conforme apontado por [Hopcroft, Ullman e Motwani 2003] e [Menezes 2011], o estudo de linguagens formais é um pilar fundamental na teoria da computação, sendo indispensável para o desenvolvimento de compiladores, analisadores léxicos e linguagens de programação. As gramáticas formais são responsáveis por definir as estruturas sintáticas válidas dessas linguagens, e sua correta formulação é essencial para garantir a conformidade e a precisão dos processadores de linguagem. A implementação prática desses conceitos, como proposta neste projeto, permite consolidar conhecimentos teóricos e aplicá-los de maneira concreta em problemas computacionais relevantes.

2 Objetivos

O objetivo principal deste projeto foi desenvolver um programa em *linguagem C* capaz de reconhecer e validar gramáticas formais descritas em um arquivo de texto, de acordo com as regras definidas na gramática. O programa foi projetado para realizar as seguintes funções:

- Verificar se as regras de produção da gramática estão em conformidade com as especificações formais estabelecidas.
- Identificar e reportar erros de sintaxe, como regras malformadas ou caracteres inválidos.
- Determinar o símbolo inicial (raiz) da gramática.
- Exibir as regras válidas e o símbolo inicial identificado, caso a gramática seja considerada válida.

3 Justificativa

De acordo com [Sipser 2007] e [Sudkamp 2007], o estudo de gramáticas formais e autômatos é essencial para diversas aplicações em computação, como a análise de código-fonte e a implementação de compiladores. A validação de gramáticas formais é um dos primeiros passos na construção de sistemas que interpretam ou compilam linguagens formais, além de ser crucial para o desenvolvimento de autômatos que reconhecem essas linguagens. Esses conceitos fornecem a base teórica e prática para tecnologias amplamente utilizadas na computação moderna.

Neste contexto, a aplicação prática deste projeto não se limita ao aprendizado dos conceitos de gramáticas formais e autômatos, mas também serve como um alicerce para o desenvolvimento de ferramentas essenciais, como compiladores e analisadores léxicos, que são fundamentais na indústria de software.

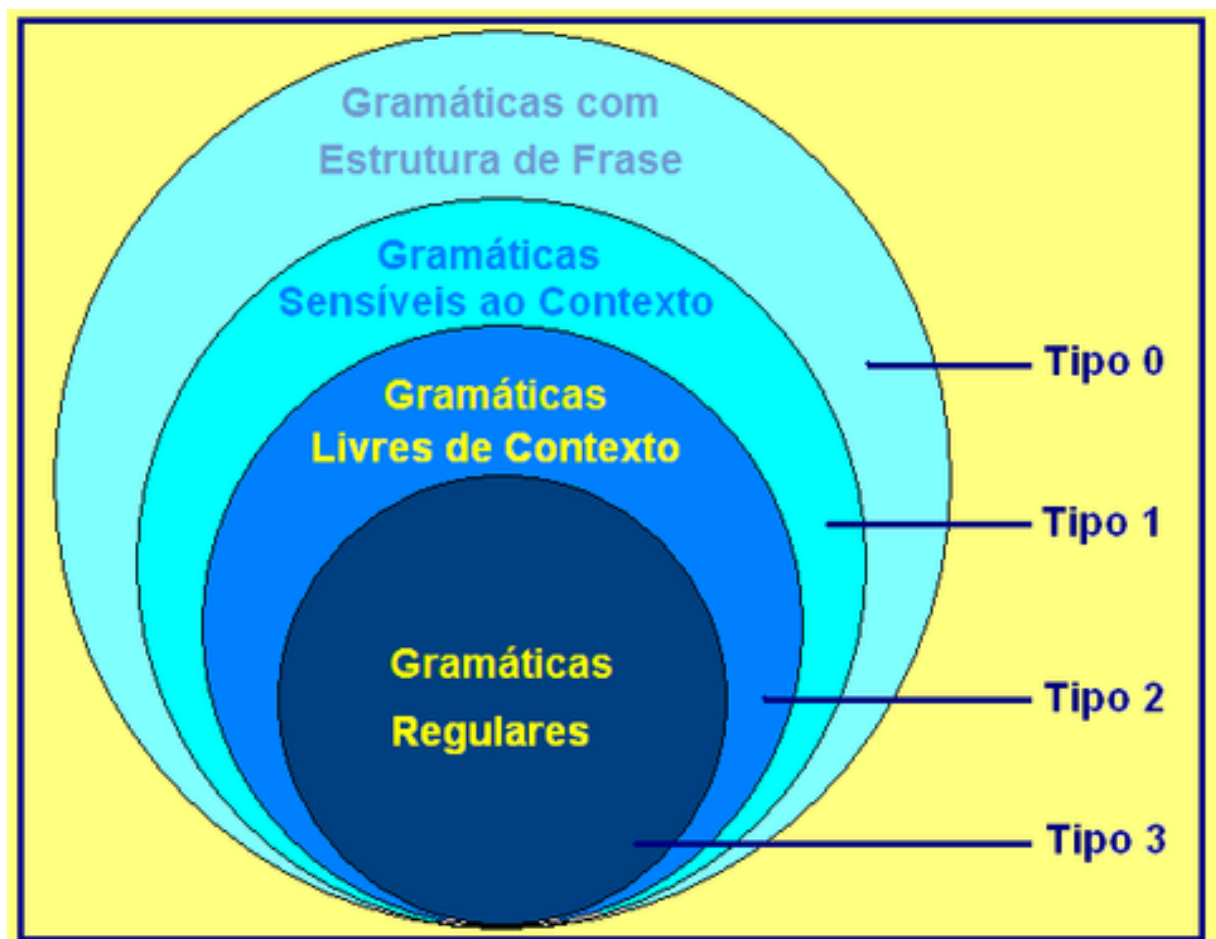
4 Fundamentação Teórica

4.1 Linguagens Formais e Gramáticas

Uma **linguagem formal** é definida como um conjunto de cadeias (ou strings) de símbolos, que são geradas a partir de um conjunto de regras de produção. Essas cadeias podem representar palavras, sentenças ou expressões, todas seguindo uma estrutura rigorosamente definida. As linguagens formais são fundamentais em várias áreas da ciência da computação, especialmente na construção de linguagens de programação, onde é essencial estabelecer a sintaxe e a semântica de maneira precisa.

A definição de uma linguagem formal é realizada por meio de uma *gramática formal*, que consiste em um conjunto de regras de produção (também conhecidas como regras de reescrita ou transformações). Essas regras determinam como as cadeias de símbolos podem ser geradas e manipuladas. As gramáticas formais são classificadas em quatro tipos principais, conforme a **Hierarquia de Chomsky**, sendo uma delas a *Gramática de Tipo 2*, ou *Gramática Livre de Contexto (GLC)*, amplamente utilizada na construção de compiladores e em diversas aplicações da teoria da computação.

Figura 1: Hierarquia de Chomsky



Fonte: [Wikipedia 2025]

4.2 Componentes de uma Gramática Formal

Uma **gramática formal** é composta por quatro componentes essenciais:

- **Não Terminais (N):** São símbolos que podem ser substituídos por outras cadeias de símbolos, formando a estrutura da linguagem. Geralmente, são representados por letras maiúsculas (A, B, C, ... Z).
- **Terminais (Σ):** São os símbolos que aparecem nas cadeias geradas pela gramática, ou seja, são os "elementos finais" da linguagem. Normalmente, são representados por letras minúsculas (a, b, c, ... z).
- **Regras de Produção (P):** Definem como os símbolos não terminais podem ser substituídos por sequências de terminais e não terminais. Cada regra é da forma:

$$A \rightarrow \alpha$$

onde $A \in N$ e $\alpha \in (N \cup \Sigma)$, ou seja, α é uma sequência de símbolos que pode conter tanto terminais quanto não terminais.

- **Símbolo Inicial (S):** Representa o ponto de partida das produções e é um elemento pertencente ao conjunto N de não terminais.

4.3 Gramáticas Livres de Contexto (GLC)

As **gramáticas livres de contexto (GLC)** são um tipo especial de gramática formal, nas quais as regras de produção seguem a seguinte estrutura:

$$A \rightarrow \alpha$$

Onde:

- **A** é um símbolo não terminal.
- α é uma sequência de símbolos, que pode incluir tanto terminais quanto não terminais.

As GLCs são fundamentais na definição da sintaxe de linguagens de programação e outras linguagens formais, sendo um pilar essencial no desenvolvimento de compiladores, interpretadores e sistemas de análise sintática. Elas são amplamente utilizadas para descrever a estrutura gramatical dessas linguagens e são cruciais para a construção de ferramentas de software que realizam a análise e a tradução de código.

4.4 Autômatos e Linguagens Formais

Os **autômatos** são modelos matemáticos de máquinas que podem reconhecer ou gerar linguagens formais. Eles desempenham um papel central na teoria das linguagens formais, fornecendo uma maneira de caracterizar e classificar diferentes tipos de linguagens. Existem vários tipos de autômatos, incluindo:

- **Autômatos Finitos:** São usados para reconhecer linguagens regulares, ou seja, linguagens que podem ser descritas por expressões regulares.

- **Autômatos de Pilha (PDA):** São utilizados para reconhecer linguagens livres de contexto (GLC), que incluem uma gama mais complexa de linguagens que as regulares.
- **Máquinas de Turing:** São capazes de reconhecer linguagens recursivamente enumeráveis, o que abrange uma classe mais ampla e poderosa de linguagens que as GLCs.

Esses autômatos são fundamentais para a teoria da computação, pois oferecem os modelos computacionais necessários para entender a capacidade de diferentes sistemas em processar e reconhecer linguagens.

4.5 Validação de Gramáticas

A **validação de gramáticas** refere-se ao processo de garantir que uma gramática esteja bem formada de acordo com as regras específicas do seu tipo. No caso das *gramáticas livres de contexto* (GLC), a validação envolve verificar:

- Que todas as produções estão corretamente definidas.
- Que o lado esquerdo de cada produção contém, no mínimo, um símbolo não terminal.
- Que o lado direito das produções está estruturado de maneira válida, ou seja, uma sequência de símbolos que pode incluir tanto terminais quanto não terminais.

Esse processo é essencial para garantir que a gramática seja bem construída e que possa ser utilizada em ferramentas de análise sintática, como compiladores e interpretadores. A validação da gramática assegura que as linguagens formais descritas por essas gramáticas possam ser processadas de forma correta e eficiente.

5 Metodologia

5.1 Estrutura do Programa

O programa foi desenvolvido em **linguagem C** e seguiu a seguinte estrutura:

- **Funções de classificação:** Para identificar se um caractere é um terminal ou não terminal.
- **Funções de validação:** Para garantir que as regras de produção da gramática estejam corretas. Isso inclui a validação dos lados esquerdo e direito de cada regra.
- **Funções auxiliares:** Como a busca pela raiz da gramática e a exibição das regras e mensagens de erro.

5.2 Validação das Regras

Cada regra foi dividida entre o **lado esquerdo** e o **lado direito**:

- O **lado esquerdo** deve conter ao menos um **não terminal**.
- O **lado direito** pode ser composto por **terminais**, **não terminais** ou até mesmo ser **vazio**.

O programa validou as regras, e, caso encontrasse algum erro (como caracteres inválidos ou estrutura incorreta), o erro seria informado ao usuário.

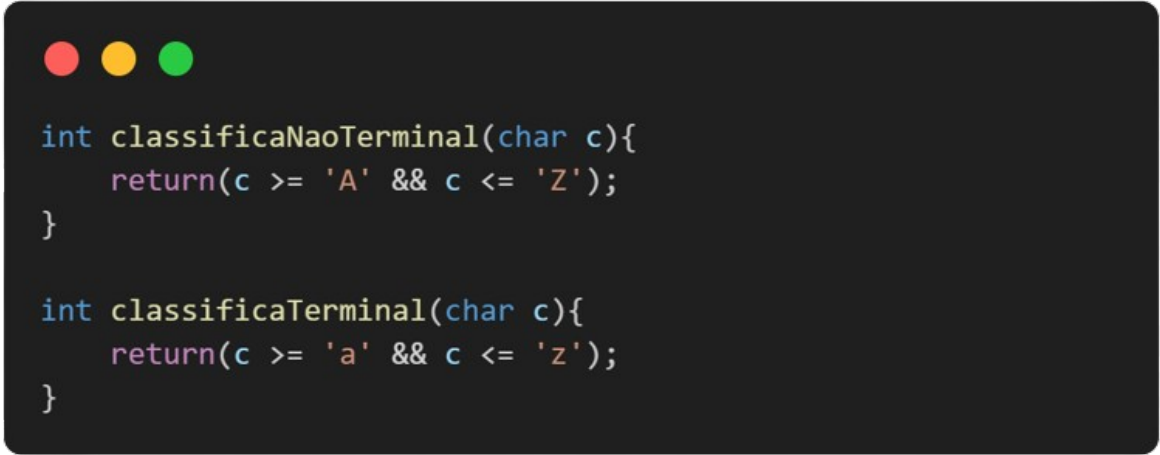
6 Desenvolvimento e Implementação

O programa foi desenvolvido em linguagem C e estruturado em funções responsáveis por diferentes tarefas:

6.1 Classificação de Símbolos

As funções '*classificaNaoTerminal*' e '*classificaTerminal*' verificam se um caractere pertence ao conjunto de terminais (A-Z) ou não terminais (a-z).

Figura 2: Funções: Classifica Não Terminal e Classifica Terminal



```
int classificaNaoTerminal(char c){  
    return(c >= 'A' && c <= 'Z');  
}  
  
int classificaTerminal(char c){  
    return(c >= 'a' && c <= 'z');  
}
```

Fonte: Autor

6.2 Validação das Regras

As funções '*valida_ladoEsq*' e '*valida_ladoDir*' têm como objetivo garantir a integridade das regras de produção, validando tanto o lado esquerdo quanto o lado direito de cada regra. Estas funções são fundamentais para assegurar que as regras estejam em conformidade com a estrutura da gramática formal, permitindo que a gramática seja processada corretamente.

A validação ocorre de acordo com os seguintes critérios:

- **Lado esquerdo:** Deve conter pelo menos um não terminal, assegurando que a regra de produção tenha um símbolo de partida, o qual representa uma variável ou não terminal que pode ser expandida em outras produções.
- **Lado direito:** Pode ser composto por uma combinação de terminais (símbolos da linguagem) e não terminais (variáveis que podem ser expandidas). Além disso, o lado direito pode ser vazio, o que caracteriza uma produção de "absorção" ou *epsilon-production*, onde o não terminal é substituído por uma cadeia vazia.
- **Símbolos inválidos:** Qualquer símbolo não reconhecido pela gramática, ou que fuja das especificações previstas, será imediatamente reportado como erro, garantindo a consistência e a correção da gramática.

Essa validação é essencial para evitar a introdução de regras malformadas, que poderiam comprometer a análise ou a execução de algoritmos baseados na gramática, como os de análise sintática e de geração de linguagens.

Figura 3: Funções: valida_ladoEsq e valida_ladoDir

```
//Funcao para verificar o lado esquerdo
int valida_ladoEsq(char* str){
    int nao_terminal = 0;

    if(str[0] == '\\0') {
        printf("\\nErro: Lado esquerdo vazio!");
        return 0;
    }

    for(int i = 0; str[i] != '\\0'; i++) {
        if(!classificaNaoTerminal(str[i])) {
            //se o caractere nao for um nao-terminal
            if(!classificaTerminal(str[i])) {
                //se o caractere nao for um terminal
                printf("\\nErro: Caractere invalido %c
encontrado na cadeia!", str[i]);
                return 0;
            }
        } else {
            nao_terminal = 1;
        }
    }
    return nao_terminal;
    //deve ter pelo menos um nao terminal
}

int valida_ladoDir(char *str){
    for(int i = 0; str[i] != '\\0'; i++) {
        if(!classificaNaoTerminal(str[i])) {
            if(!classificaTerminal(str[i])) {
                return 0; // Caractere invalido
            }
        }
    }
    return 1;
}
```

Fonte: Autor

6.3 Identificação da Raiz

A função '*encontraRaiz*' tem como objetivo identificar o primeiro não terminal presente na gramática, que é considerado o símbolo inicial ou "raiz" da gramática. Este símbolo serve como ponto de partida para a derivação das produções na análise sintática.

6.4 Processamento da Gramática

A função '*valida_gramatica*' realiza o processamento da gramática de acordo com os seguintes passos:

1. **Remove o caractere \$:** O caractere \$ é removido do final da cadeia, representando o símbolo de terminação da gramática.
2. **Separa a string em regras:** A gramática fornecida é dividida em regras de produção, cada uma representando uma relação entre um símbolo não terminal e seus possíveis expansões.
3. **Valida cada regra individualmente:** Cada regra é analisada para garantir que esteja corretamente formada, conforme as definições da gramática.
4. **Armazena as regras válidas em um vetor dinâmico:** As regras válidas são armazenadas de forma eficiente em um vetor dinâmico, permitindo fácil manipulação e consulta.
5. **Identifica a raiz da gramática:** A função '*encontraRaiz*' é invocada para identificar o símbolo inicial da gramática.

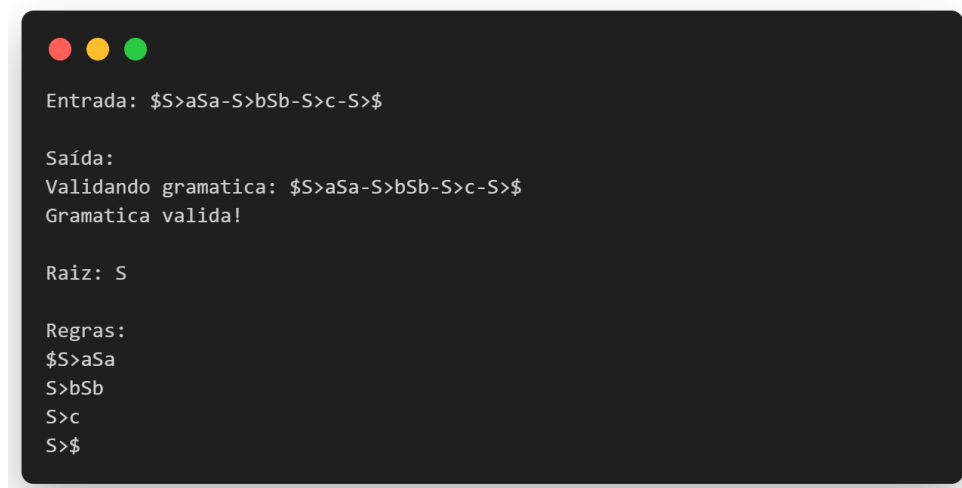
Execução Principal O programa principal (*main*) realiza as seguintes operações:

- Lê a gramática de um arquivo fornecido como argumento de linha de comando.
- Invoca a função de validação para garantir que a gramática esteja no formato correto.
- Exibe os resultados, incluindo a identificação da raiz e as regras válidas.

Exemplo de execução:

```
$ ./reconhecedor entrada.txt
```

Figura 4: Execução do programa com gramática válida



```
Entrada: $S>aSa-S>bSb-S>c-S>$

Saída:
Validando gramatica: $S>aSa-S>bSb-S>c-S>$
Gramatica valida!

Raiz: S

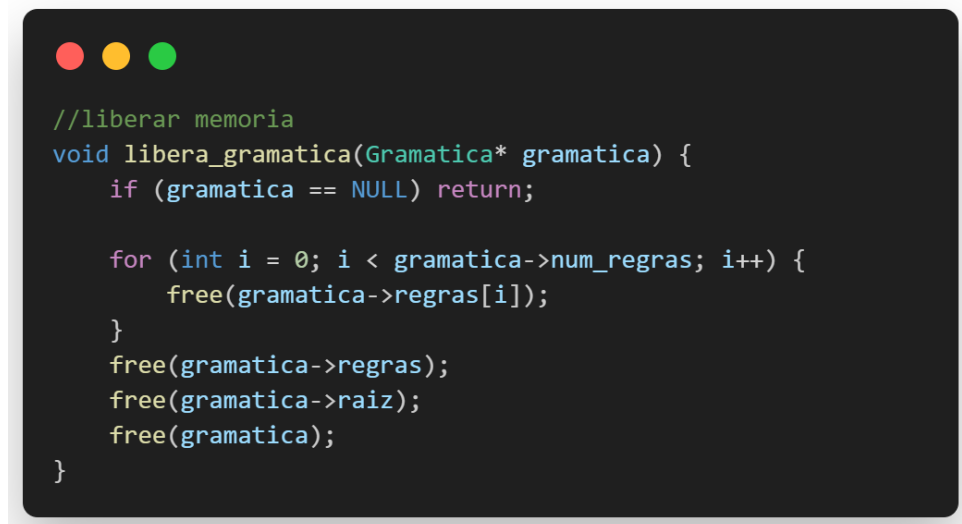
Regras:
$S>aSa
S>bSb
S>c
S>$
```

Fonte: Autor

6.5 Liberação de Memória

A função *libera_gramatica* libera toda a memória alocada dinamicamente para evitar vazamentos (memory leaks).

Figura 5: Função: libera_gramatica



```
//liberar memoria
void libera_gramatica(Gramatica* gramatica) {
    if (gramatica == NULL) return;

    for (int i = 0; i < gramatica->num_regras; i++) {
        free(gramatica->regras[i]);
    }
    free(gramatica->regras);
    free(gramatica->raiz);
    free(gramatica);
}
```

Fonte: Autor

7 Conclusão

O projeto **Reconhecedor de Gramáticas** foi desenvolvido com sucesso, alcançando seu principal objetivo de validar automaticamente gramáticas formais representadas por regras de produção. Com uma implementação modular e eficiente, o programa provou ser capaz de identificar a estrutura sintática correta de uma gramática e de sinalizar erros de forma precisa.

A experiência permitiu aplicar conhecimentos teóricos na prática, consolidando a importância das gramáticas no processo de compilação e construção de linguagens. Além disso, o desenvolvimento em linguagem C reforçou práticas de manipulação de strings, alocação dinâmica e estruturação modular.

Referências

HOPCROFT, J. E.; ULLMAN, D. J.; MOTWANI, R. *Introdução à teoria de autômatos, linguagens e computação*. 2. ed. Rio de Janeiro: Campus, 2003.

MENEZES, P. B. *Linguagens formais e autômatos*. 6. ed. Porto Alegre: Artmed, 2011.

SIPSER, M. *Introdução à teoria da computação*. 2. ed. São Paulo: Thomson Learning, 2007.

SUDKAMP, T. A. *Languages and machines: an Introduction to the theory of computer science*. 3. ed. India: Pearson Education, 2007.

WIKIPEDIA. *Hierarquia de Chomsky*. 2025. 02/06/2025. Disponível em: https://pt.wikipedia.org/wiki/Hierarquia_de_Chomsky?oldformat=true.