

Reconhecedor de Gramáticas Regulares

Arthur Pereira do Santos (m81375)

Fernando da Costa Kudrna (m78181)

Yan Rodrigues dos Santos (m13938)

Disciplina de Linguagens Formais

Departamento de Informática – UNISC

Campus de Santa Cruz do Sul

96.810-206 - Santa Cruz do Sul - RS - Brasil

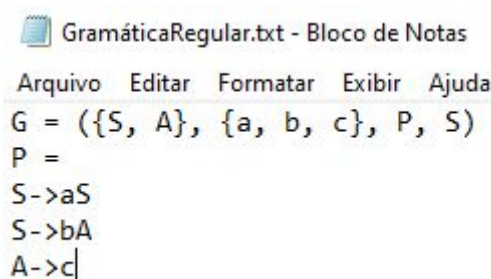
arthur1@mx2.unisc.br, fernandokudrna@mx2.unisc.br, yan6@mx2.unisc.br

1. Sobre o projeto

Desenvolvido em Java, o aplicativo é capaz de reconhecer uma gramática regular, verificar se a gramática contém erros (diferenciando quando não se tratar de uma gramática regular) e gerar sentenças, demonstrando o caminho feito nos símbolos Não-terminais para chegar a tal sentença.

2. Modo de usar

Inicialmente, deve-se inserir o caminho do arquivo de texto a ser usado para a identificação da gramática escolhida. A entrada(dentro do arquivo), deve seguir o seguinte padrão:



```
GramáticaRegular.txt - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
G = ({S, A}, {a, b, c}, P, S)
P =
S -> aS
S -> bA
A -> c|
```

Imagem 1. Formato padrão da inserção da entrada.

Como pode ser observado, não-terminais que podem levar a mais de um caminho devem ser duplicados para que o programa reconheça de forma correta.

```

public static void main(String[] args) throws IOException {
    String path = "C:\\Users\\arthur\\Desktop\\Projeto_LF-master\\GramáticaRegular.txt";
    LeituraArquivo.leitor(path);
}

```

Imagem 2. Inserção do arquivo via código.

Iniciando a execução do programa, o software vai processar a entrada presente no arquivo, identificando os terminais, não-terminais e regras de produção, separando-os em várias variáveis auxiliares.

```

G = ({S, A}, {a, b, c}, P, S)
{S, A}, {a, b, c}, P, S
P =
S->aS
S->bA
A->c
Nao Terminais: S, A
Terminais: a, b, c
Simbolo inicial da gramática: S
[S->aS, S->bA, A->c]

```

Imagem 3. Processamento da entrada inserida via arquivo.

```

public static void identgram(String[] a) {
    //variáveis
    List naoterminais = new ArrayList();
    List terminais = new ArrayList();
    List ini = new ArrayList();
    ArrayList<String> prod = new ArrayList<>();
    //atribuição do valor para variáveis
    String term = "";
    String naoterm = "";
    String inicial = "";
    int cont = 1;
    int i = 0;
    if (i == 0) { //identifica qual linha estamos no arquivo
        System.out.println(a[i]);
        String result = a[i].substring(a[i].indexOf("(") + 1, a[i].indexOf(")"));
        System.out.println(result);
        for (int j = 0; j < result.length(); j++) {
            char c = result.charAt(j);
            if (c == '(') {
                do {
                    if (c != ')' & c != ',' & c != '{' & c != ' ' & cont == 1) //verificar se é letra e está na primeira {}
                    {
                        naoterminais.add(c); //adiciona na list de Nao terminais
                    }
                    if (c != ')' & c != ',' & c != '{' & c != ' ' & cont == 2) //verificar se é letra e esta na segunda {}
                    {
                        terminais.add(c); //adiciona na list de terminais
                    }
                    j++; //nao parar o for
                    c = result.charAt(j); //verificar o que tem na String
                } while (c != ')');
                cont++;
                if (cont == 3) {
                    ini.add(result.charAt(result.length() - 1)); //verifica o simbolo de inicialização
                }
            }
        }
    }
}

```

Imagem 4. Código usado para separação das partes da gramática, armazenando-as nas suas respectivas estruturas auxiliares.

Logo após o processamento da entrada, o software identifica se se trata ou não de uma Gramática Regular (GR) usando as regras descritas para as regras de produção e exibe na tela.

```
G = ({S, A}, {a, b, c}, P, S)
{S, A}, {a, b, c}, P, S
P =
S->aS
S->bA
A->c
Nao Terminais: S, A
Terminais: a, b, c
Simbolo inicial da gramática: S
[S->aS, S->bA, A->c]
Gramática é Regular
```

Imagem 5. Exemplo de identificação de Gramática Regular.

```
G = ({S, A}, {a, b, c}, P, S)
{S, A}, {a, b, c}, P, S
P =
S->aS
S->bAA
A->c
Nao Terminais: S, A
Terminais: a, b, c
Simbolo inicial da gramática: S
[S->aS, S->bAA, A->c]
Gramática Não é Regular
BUILD SUCCESSFUL (total time: 0 seconds)
```

Imagem 6. Exemplo da identificação de que não se trata de uma Gramática Regular.

```

static boolean detectgr(ArrayList<String> prods, String nt, String t) {
    for (String prod : prods) {
        String[] full = prod.split("->");//separa o nao terminal das suas producoes
        String esq = full[0];//onde fica o nao terminal
        String dir = full[1];//as producoes
        String remont = "";

        for (int i = 0; i < dir.length(); i++) {
            String aux = "" + dir.charAt(i);
            if (aux.equals("/")) {
                remont = remont + "99";
            } else {
                remont = remont + aux;
            }
        }
        String[] fulldir = remont.split("99");//pega os valores que cada producao
        //verifica se gramatica e regular
        if (esq.length() > 1 || !nt.contains(esq)) {
            return false;
        }
        for (int i = 0; i < fulldir.length; i++) {
            if (fulldir[i].length() > 2) {
                return false;
            }
            if (fulldir[i].length() == 1 && !t.contains(fulldir[i])) {
                return false;
            }
            if (fulldir[i].length() == 2) {
                String aux = "" + fulldir[i].charAt(0);
                if (!t.contains(aux)) {
                    return false;
                }
                aux = "" + fulldir[i].charAt(1);
                if (!nt.contains(aux)) {
                    return false;
                }
            }
        }
    }
    return true;
}

```

Imagem 7. Código referente a identificação da Gramática Regular.

Por fim, é solicitado ao usuário o número de sentenças de exemplo(que a gramática produz de acordo com suas regras de produção) que deseja exibir na tela. Quando o usuário define um número, o algoritmo começa a criar sentenças aleatoriamente e exibe o caminho, nas regras de produção, que o algoritmo tomou para chegar a cada sentença que ele exibiu.

```

G = ({S, A}, {a, b, c}, P, S)
{S, A}, {a, b, c}, P, S
P =
S->aS
S->bA
A->c
Gramática é Regular
Nao Terminais: S, A
Terminais: a, b, c
Símbolo inicial da gramática: S
[S->aS, S->bA, A->c]
Número de sentenças que o usuário quer:
5
1ª Sentença: [b, c]
Caminho de nao terminais usados na 1 tentativa: [S, A]
2ª Sentença: [a, b, c]
Caminho de nao terminais usados na 2 tentativa: [S, S, A]
3ª Sentença: [b, c]
Caminho de nao terminais usados na 3 tentativa: [S, A]
4ª Sentença: [a, b, c]
Caminho de nao terminais usados na 4 tentativa: [S, S, A]
5ª Sentença: [a, b, c]
Caminho de nao terminais usados na 5 tentativa: [S, S, A]
BUILD SUCCESSFUL (total time: 1 second)

```

Imagem 8. Possibilidade do usuário escolher quantas sentenças o programa deve rodar e execução do programa em geral.

```

//copular vetores com producoes e nao term
for (String prod : prods) {
    String[] full = prod.split(">"); //separa o nao terminal das suas producoes
    nterm[i] = full[0]; //onde fica o nao terminal
    p[i] = full[1]; //onde fica as producoes
    i++;
}
System.out.println("Número de sentenças que o usuário quer: ");
n = ler.nextInt();

for (int k = 1; k <= n; k++) {
    sentencas.clear();
    nt2.clear();

    count2 = 0;
    String iniaux = ini; //String para aux no uso dos nao terminais
    nt2.add(iniaux);
    while (count2 <= 10 && iniaux != null) { //limite para +- 10 o tamanho das sentenças
        aleataux = "";
        aleat = (term.charAt(r.nextInt(term.length()))); //seleciona um terminal aleatorio como objetivo
        aleataux += aleat;

        valorinalcancavel = false;
        valoralcancado = false;

        while (valoralcancado == false & valorinalcancavel == false || valoralcancado == true & valorinalcancavel == false & valoralcancado == false & valorinalc
            count = 0;
            tnaux = "";
            tn = (termounao.charAt(r.nextInt(termounao.length()))); //1 = procurar terminal sozinho, 2 = terminal com nao terminal
            tnaux += tn;

            if ("1".equals(tnaux)) {
                for (int j = 0; j < prods.size(); j++) {
                    if (nterm[j].equals(iniaux)) {
                        if (p[j].length() == 1 && p[j].equals(aleataux) == true) { //verifica o tamanho da producao e se o terminal objetivo se encontra nela
                            valoralcancado = true; //informa q terminal objetivo foi alcançado e nao tem mais nao terminal na sentença
                            sentencas.add(aleataux); //add o terminal alcançado na sentença
                            iniaux = null;
                            count2++;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

