Analisador Preditivo Tabular

Arthur G. Q. Kunzler[1], Guilherme M. Dornelles[1] Vinícius R. Boff[1]

[1]Engenharia de Software - Pontifícia Universidade Católica do Rio Grande do Sul(PUCRS) - Porto Alegre - RS - Brasil

{arthur.kunzler, guilherme.dornelles, vinicius.rech} @edu.pucrs.br

1. Introdução

Este relatório consiste no desenvolvimento do Trabalho Prático 2, da disciplina de Linguagens de Programação, turma do segundo semestre de 2022.

Como objetivo do trabalho, era necessário implementar um Analisador Preditivo Tabular como parte da etapa de Análise Sintática no estudo de Compiladores. O grupo escolheu desenvolver a implementação em Java, utilizando o paradigma Orientado a Objetos.

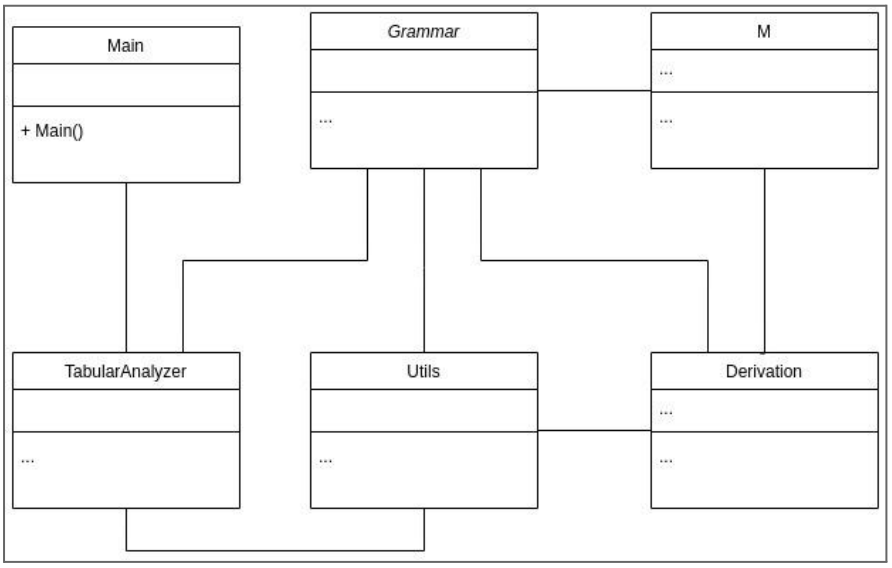
O desenvolvimento foi dividido em três etapas: primeiramente, leitura de uma Gramática, não-ambígua e sem recursão à esquerda, de um arquivo texto; segunda etapa, definição dos conjuntos First e Follow da Gramática; por último, gerar a Tabela de Análise Preditiva. Sendo que, as etapas dois e três devem ser mostradas na tela.

No decorrer do trabalho, é descrito detalhes de cada uma das partes.

2. Classes utilizadas e suas principais funcionalidades

Para a implementação da solução, algumas classes foram criadas para auxiliar na abstração e encapsular responsabilidades individualmente. Na figura 1 pode ser visto de uma maneira simplificada como ficou o diagrama de classes da nossa aplicação.

Figura 1: Diagrama de classes construído durante o trabalho.



Fonte: De autoria própria

S

A classe Main serve para rodar a aplicação, a classe Utils serve para alguns métodos auxiliares. A classe TabularAnalyzer é a controladora geral do programa, ela possui o papel de gerenciar o uso da classe Grammar que representa uma abstração da Gramática, e por consequência também controla as instâncias da classe Derivation. Essa, por sua vez, representa uma derivação da gramática, possuindo o lado Esquerdo (não-terminal) e o lado direito da gramática, suas produções. Por último, a classe M é a abstração de uma relação da tabela. Ela possui o terminal e o não-terminal que levam a uma determinada produção (derivação). É usada na classe *Grammar* como uma lista que representa as possibilidades da Tabela de Análise.

3. Desenvolvimento da Solução

3.1. Utilizando a Aplicação

Para fazer uso da aplicação, é necessário iniciar pelo método *main* da classe Main. Ele é o ponto inicial do programa. Pode-se passar por linha de comando um argumento contendo o caminho do arquivo texto que contém a gramática, mas fazer isso é opcional, pois por padrão, o programa utiliza um arquivo de caminho *input.txt*. A partir do método *main*, há a instalação da classe necessária e chamadas dos métodos que irão mostrar na tela o que é preciso.

3.2. First

Para encontrar o First, um método percorre individualmente cada derivação e a partir do primeiro caractere de cada sentença verifica se o mesmo é sentença vazia, se sim, adiciona-o à uma lista auxiliar que será usada posteriormente. Caso contrário, verifica se o caractere é terminal, se sim, adiciona-o à lista auxiliar, se não, chama recursivamente a própria função passando a derivação relacionada ao caractere não terminal que havia sido encontrada e o retorno da função é adicionado à lista auxiliar. Finalmente, o conjunto First é o retorno recursivo desse método.

3.3. Follow

Após a construção do processo de análise do First de cada símbolo não terminal, o programa realiza a análise do Follow deles. Para isso seguimos as seguintes regras: Caso o símbolo for o primeiro da gramática, adicionamos “$” ao seu conjunto de Follow. Agrupamos todas as derivações onde o símbolo não terminal analisado aparece e, dependendo caso o símbolo for seguido de outro símbolo terminal ou não, adicionamos o próprio símbolo seguinte ou os símbolos terminais do seu conjunto First, respectivamente. Caso o símbolo for o último caractere da derivação, adicionamos o conjunto Follow do seu símbolo gerador ao seu conjunto Follow.

3.4. Analisador Preditivo Tabular

Após o encontro dos conjuntos First e Follow, a organização da Tabela de Análise foi possível. Ela acontece, criando uma lista de instâncias de *M*.

Para a criação dos itens da lista, o método *buildAnalysisTable()* é o responsável por isso. Nele, faz-se um *loop* em todas as produções possíveis da Gramática, buscando os First de cada uma, e adicionando *M[não-terminal, terminal]*

para cada terminal achado indicando a produção em questão. Caso seja encontrada a palavra vazia (“E”) no First, adiciona-se também o Follow do não-terminal à tabela.

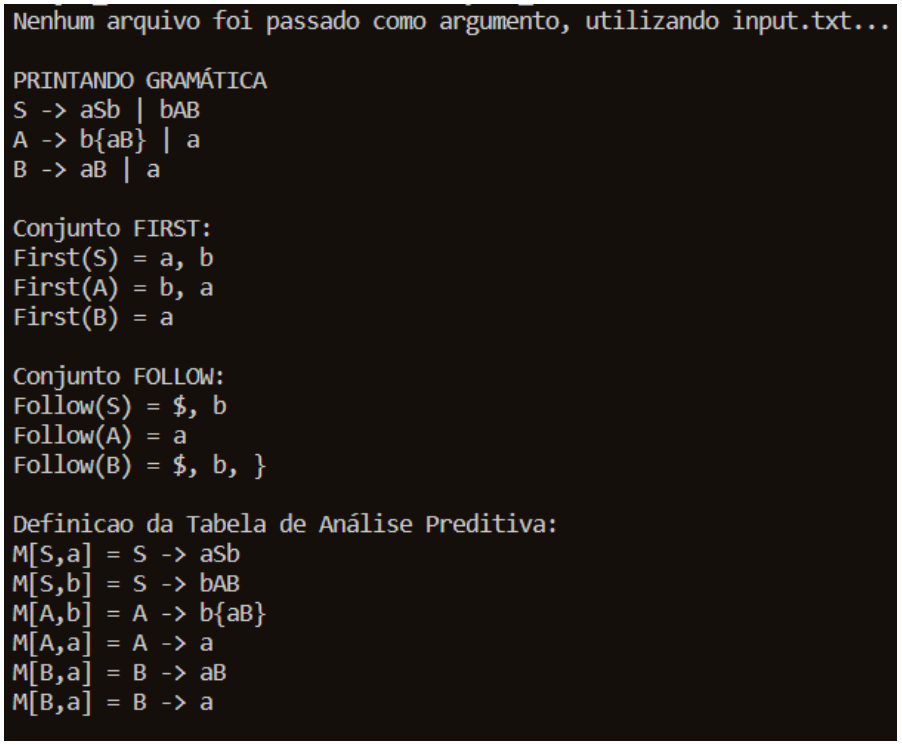
Ao final, exibe na tela todas as instâncias criadas, gerando um output como pode ser visto nas Figuras a seguir, com exemplos e testes de execução:

Figura 2: Output de teste completo do programa executado

```
Nenhum arquivo foi passado como argumento, utilizando input.txt...

PRINTANDO GRAMÁTICA
A -> TA'
A' -> +TA' | E
T -> FT'
T' -> *FT' | E
F -> (A) | i

Conjunto FIRST:
First(A) = (, i
First(A') = +, E
First(T) = (, i
First(T') = *, E
First(F) = (, i

Conjunto FOLLOW:
Follow(A) = $, )
Follow(A') = $, )
Follow(T) = +, $, )
Follow(T') = +, $, )
Follow(F) = *, +, $, )

Definicao da Tabela de Análise Preditiva:
M[A,(] = A -> TA'
M[A,i] = A -> TA'
M[A',+] = A' -> +TA'
M[A',$] = A' -> E
M[A',)] = A' -> E
M[T,(] = T -> FT'
M[T,i] = T -> FT'
M[T',*] = T' -> *FT'
M[T',+] = T' -> E
M[T',$] = T' -> E
M[T',)] = T' -> E
M[F,(] = F -> (A)
M[F,i] = F -> i
```

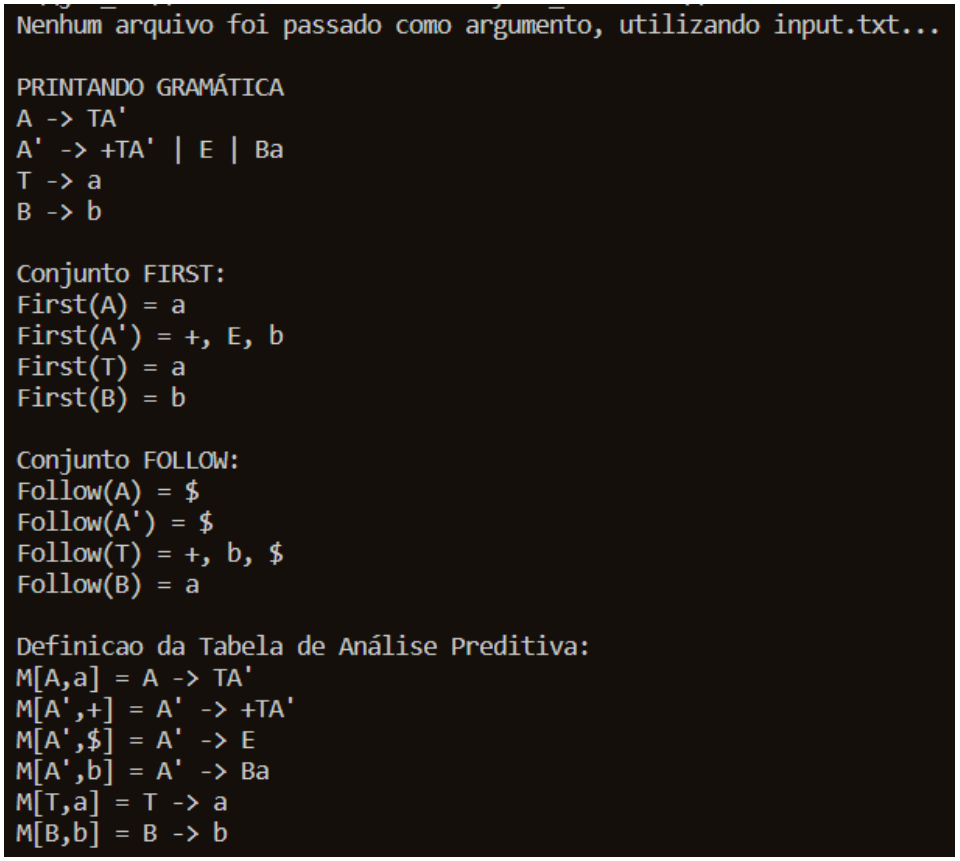
Fonte: De autoria própria

Figura 3: Output com outro teste da execução



Fonte: De autoria própria

Figura 4: Output completo com outra gramática de teste



Fonte: De autoria própria