# Analisador Preditivo não-recursivo

Matéria: Linguagens de Programação

Integrantes: Gabriel Spiandorello e Léo Falcão

Professor(a): Andréa Aparecida Konzen

## Introdução

Neste trabalho prático, foi desenvolvido um Analisador Preditivo Tabular para realizar a análise sintática de uma gramática. O objetivo era verificar se a gramática era LL e, caso fosse, simular a análise da gramática passo a passo, demonstrando o reconhecimento de uma sentença.

## Planejamento

O desenvolvimento do Analisador Preditivo Tabular seguiu os seguintes passos:

- Recebimento da gramática: a gramática foi recebida como entrada e testada para verificar se era LL.
- Criação do conjunto FIRST: para verificar se a gramática era LL, foi necessário criar o conjunto FIRST.
- Criação do conjunto FOLLOW: após a criação do conjunto FIRST, foi necessário criar o conjunto FOLLOW.
- Montagem da tabela de análise: com a criação dos conjuntos FIRST e FOLLOW, foi possível montar a tabela de análise preditiva tabular.

## Implementação

A implementação do Analisador Preditivo Tabular foi realizada na linguagem de programação Java. Para a criação da tabela de análise, foram utilizadas estruturas de dados como Arrays.

## Resultados

Durante a etapa de desenvolvimento, realizamos a verificação da gramática para LL, geramos os conjuntos FIRST, o que nos proporcionou uma compreensão mais profunda da gramática em questão.

No entanto, não conseguimos ainda implementar o FOLLOW, pois enfrentamos dificuldades em utilizar a estrutura de dados selecionada, e tampouco conseguimos executar a verificação de sentenças, funcionalidades essenciais para a análise sintática da gramática. Essas etapas que não foram implementadas ficam como trabalhos futuros para aprimorar nosso conhecimento sobre análise sintática.

Como não conseguimos gerar os símbolos FOLLOW, nós decidimos inserir eles manualmente, para ser possível gerar a tabela de análise preditiva tabular. Tivemos êxito na geração da tabela e abaixo apresentamos algumas fotos do código implementado, como o método addFirst, verificaGramaticaLL, gerarTabela.

# Testes

**Gramática a ser testada:**

G (L) = ({S, U, T, V, F }, { +, *, (, ), id }, P, S)

P: { S -> TU
U -> +TU | E
T -> FV
V -> *FV | E
F -> ( E ) | id }

**Resultados Esperados:**

**First e Follow:**

First(S) = { (, id }
First(U) = { +, E }
First(T) = { (, id }
First(V) = { *, E }
First(F) = { (, id }

Follow(S) = { $, ) }
Follow(U) = { $, ) }
Follow(T) = { $, +, ) }
Follow(V) = { $, +, ) }
Follow(F) = { *, +, ), $ }

**Geração da Tabela:**
S ⇒ TU ... FIRST(TU) = { (, id }
Adicionar S ⇒ TU em M[E,(] ; M[E,id]

U ⇒ +TU ... FIRST(+TU) = { + }
Adicionar U ⇒ +TU em M[U,+]

U ⇒ E ... FOLLOW(U) = { $, ) }
Adicionar U ⇒ E em M[U,$] ; M[U,)]

T ⇒ FV ... FIRST(FV) = { (, id }
Adicionar T ⇒ FV em M[T,(] ; M[T,id]

V ⇒ *FV ... FIRST(*FV) = { * }
Adicionar V ⇒ *FV em M[V,*]

V ⇒ E ... FOLLOW(V) = { +,),$ }
Adicionar V ⇒ E em M[V,+] ; M[V,)] ; M[V,$]

F ⇒ ( E ) ... FIRST((E)) = { ( }
Adicionar F ⇒ (E) em M[F,(]

F ⇒ id ... FIRST(id) = { id }
Adicionar F ⇒ id em M[F,id]

# Fotos do código:

```java
/*Este foi o método que realizar para adicionar o First ao símbolo passado por parâmetro */
public void addFirst(String simbolo){
    for(int i = 0; i < naoTerminais.size(); i++){
        if(simbolo.equals(naoTerminais.get(i).getNaoTerminal())){
            for(int j = 0; j < naoTerminais.get(i).getDerivacoes().size(); j++){
                String derivacao = naoTerminais.get(i).getDerivacoes().get(j);
                if(derivacao.equals(anObject:"id")){
                    naoTerminais.get(i).addFirst(derivacao);
                }
                else{
                    naoTerminais.get(i).addFirst(String.valueOf(derivacao.charAt(index:0)));
                }
            }
        }
    }
}

/*Este método chama o método addFirst para todos os símbolos não terminais */
public void addTodosFirst(){
    for(int i = 0; i < naoTerminais.size(); i++){
        addFirst(naoTerminais.get(i).getNaoTerminal());
    }
}
```

```java
/*Este método foi desenvolvido para arrumar todos os First que estão com não terminais*/
public void arrumandoFirstNaoTerminal(){
    boolean verifica = true;
    while(verifica){
        verifica = false;
        for(int i = 0; i < naoTerminais.size(); i++){
            for(int j = 0; j < naoTerminais.get(i).getFirst().size(); j++){
                for(int k = 0; k < naoTerminais.size(); k++){
                    if(naoTerminais.get(i).getFirst().get(j).equals(naoTerminais.get(k).getNaoTerminal())){
                        naoTerminais.get(i).setFirst(naoTerminais.get(k).getFirst());
                        verifica = true;
                    }
                }
            }
        }
    }
}
```

```java
/*Este foi o método que verifica se a gramática é LL, verificando se há recursão a esquerda ou disjunção par a par */
public void verificaGramaticaLL(){
    boolean temRecursaoAEsquerda = false;
    boolean temDisjuncaoParAPar = false;

    for(int i = 0; i < naoTerminais.size(); i++){
        for(int j = 0; j < naoTerminais.get(i).getFirst().size(); j++){
            String first = naoTerminais.get(i).getFirst().get(j);
            if(naoTerminais.get(i).getNaoTerminal().equals(first)){
                temRecursaoAEsquerda = true;
            }
        }
    }

    for(int i = 0; i < naoTerminais.size(); i++){
        for(int j = 0; j < naoTerminais.get(i).getFirst().size(); j++){
            String first = naoTerminais.get(i).getFirst().get(j);
            for(int k = 0; k < naoTerminais.get(i).getFirst().size(); k++){
                if(j!=k && first.equals(naoTerminais.get(i).getFirst().get(k))){
                    temDisjuncaoParAPar = true;
                }
            }
        }
    }
```

```java
    if(temRecursaoAEsquerda && temDisjuncaoParAPar){
        System.out.println(x:"\nEssa gramática apresenta recursão à esquerda e disjunção par a par");
        System.out.println(x:"Por favor, rode o programa novamente e insira outra gramática");
    }
    else if(temRecursaoAEsquerda){
        System.out.println(x:"\nEssa gramática apresenta recursão à esquerda");
        System.out.println(x:"Por favor, rode o programa novamente e insira outra gramática");
    }
    else if(temDisjuncaoParAPar){
        System.out.println(x:"\nEssa gramática apresenta disjunção par a par");
        System.out.println(x:"Por favor, rode o programa novamente e insira outra gramática");
    }
    else{
        System.out.println(x:"\nEssa é uma gramática LL, pode selecionar as outras opções do menu");
    }
}
```

```java
/*Neste método tentamos gerar a tabela com os Follows inseridos manualmente */
public void gerarTabela(){
    for(int i = 0; i < naoTerminais.size(); i++){
        for(int j = 0; j < naoTerminais.get(i).getDerivacoes().size(); j++){
            String derivacao = naoTerminais.get(i).getDerivacoes().get(j);
            if(derivacao.equals(anObject:"E")){

            }
            String print = "";
            String printDerivacoes = "";
            ArrayList<String> derivacoes = new ArrayList<String>();
            for(int k = 0; k < naoTerminais.get(i).getFirst().size(); k++){
                if(!naoTerminais.get(i).getFirst().get(k).equals(anObject:"E")){
                    if(k == naoTerminais.get(i).getFirst().size()-1){
                        printDerivacoes += naoTerminais.get(i).getFirst().get(k) + " }";
                        derivacoes.add(naoTerminais.get(i).getFirst().get(k));
                    }
                    else{
                        printDerivacoes += naoTerminais.get(i).getFirst().get(k) + ", ";
                        derivacoes.add(naoTerminais.get(i).getFirst().get(k));
                    }
                }
            }
            print += naoTerminais.get(i).getNaoTerminal() + " -> " + derivacao + " ... FIRST(" + derivacao + ") = { " +
            printDerivacoes;

            for(int l = 0; l < derivacoes.size(); l++){
                print += "\nAdicionar " + naoTerminais.get(i).getNaoTerminal() + " -> " + derivacao + " em M[E," + derivacoes.get
                (l) + "]";
            }
            System.out.println(print);
        }
    }
}
```

```
TABELA PREDITIVA TABULAR:

S -> TU ... FIRST(TU) = { (, id }
Adicionar S -> TU em M[E,(]
Adicionar S -> TU em M[E,id]
U -> +TU ... FIRST(+TU) = { +,
Adicionar U -> +TU em M[E,+]
U -> E ... FIRST(E) = { +,
Adicionar U -> E em M[E,+]
T -> FV ... FIRST(FV) = { (, id }
Adicionar T -> FV em M[E,(]
Adicionar T -> FV em M[E,id]
V -> *FV ... FIRST(*FV) = { *,
Adicionar V -> *FV em M[E,*]
V -> E ... FIRST(E) = { *,
Adicionar V -> E em M[E,*]
F -> (E) ... FIRST((E)) = { (, id }
Adicionar F -> (E) em M[E,(]
Adicionar F -> (E) em M[E,id]
F -> id ... FIRST(id) = { (, id }
Adicionar F -> id em M[E,(]
Adicionar F -> id em M[E,id]
```