**TRABALHO PRÁTICO - ANÁLISE SINTÁTICA**

Bruno Marcos Pinheiro da Silva
201565552AC

## 1. INTRODUÇÃO

Este documento visa descrever as principais características da implementação do Analisador Sintático para a linguagem "*lang*", proposta na disciplina de Teoria dos Compiladores.

O analisador foi desenvolvido com o auxílio da ferramenta ANTLR[1]. Para isso tanto as definições léxicas previamente implementadas na ferramenta JFlex quanto a gramática livre de contexto que representa as construções da linguagem foram escrita foram adaptadas para a sintaxe do ANTLR.

## 2. ESTRUTURA DE ARQUIVOS

O trabalho consiste na seguinte estrutura de arquivos:

- `lang/`
  - `ast/`
    - `SuperNode.java`
    - `Arquivos referentes aos nós da AST`
  - `parser/`
    - `TestParser.java`
    - `ParseAdaptor.java`
    - `FinalParser.java`
    - `CustomErrorListener.java`
    - `BuildAstVisitor.java`
    - `(LangLexer/LangParser.java)`
  - `testes/...`
  - `LangCompiler.java`
  - `LangLexer.g4`
  - `LangParser.g4`
  - `Compile.bash`
  - `CompileAntlr.bash`

[1] https://www.antlr.org/

## 3. COMO EXECUTAR

Para usar o programa, basta executar as seguintes linhas no terminal de comando aberto na pasta do projeto:

- O .jar do ANTLR deve estar na pasta `.\lang` do projeto.
- Para compilar as classes Java e executar o programa, a partir da pasta `lang`:

```
javac -cp .:antlr-4.8-complete.jar ast/*.java parser/*.java LangCompiler.java -d .
java -classpath .:antlr-4.8-complete.jar lang.LangCompiler -bs
```

- Caso seja necessário criar as classes do ANTLR novamente, deve-se executar os seguintes comandos:

```
java -jar ./antlr-4.8-complete.jar -o ./parser/ LangLexer.g4
java -jar ./antlr-4.8-complete.jar -o ./parser/ LangParser.g4 -visitor
```

Os arquivos `compile.bash` e `compileAntlr.bash` serão incluídos no projeto para executar esses comandos automaticamente, considerando a presença do ANTLR (`antlr-4.8-complete.jar`) na pasta `lang`.

- A pasta `./testes` possui os testes fornecidos previamente para a implementação do trabalho, sem alterações nos arquivos.

## 4. DECISÕES DE PROJETO

### 4.1. Ferramenta Utilizada

Como anteriormente citado, a ferramenta utilizada foi o ANTLR. Esta ferramenta é útil pois aceita gramáticas livres de contexto desde que não contenham recursão a esquerda de forma indireta ou “escondida” (quando uma derivação em vazio gera a recursão). Como a gramática proposta para a linguagem lang não possuía estas características, a sua implementação foi relativamente simples.

O ANTLR4 gera um parser recursivo descendente com uma função ALL(*) para predizer as produções da gramática. Esta abordagem é discutida pelo autor da ferramenta em [1], que utiliza estratégias para construir autômatos de lookahead para verificar as derivações possíveis com os inputs lidos.

### 4.2. Separação entre Léxico e Sintático

O ANTLR requer a definição dos tokens léxicos dentro da própria ferramenta. No caso deste trabalho, as definições Léxicas foram definidas em um arquivos `LangLexer.g4`, diretamente adaptadas do trabalho anterior, com as seguintes mudanças:

- uso da construção **`→skip`** nos tokens de espaço em branco, nova linha e comentários, que indicam ao lexer para ignorar estas expressões.
- uso de `fragment` para indicar expressões que não geram tokens

Já o analisador sintático foi definido no arquivo `LangParser.g4`. Esse arquivo reconhece os tokens do Lexer pela definição da opção `tokenVocab`.

Estes arquivos geram as classes `LangLexer` e `LangParser` no pacote `lang.parser` e são utilizados pela classe `LangFinal` implementada.

### 4.3. Escrita da gramática

Algumas alterações foram feitas na gramática, porém somente devido a sintaxe do ANTLR ou para facilitar etapas posteriores, e não por questões da aceitação ou corretude da gramática:

- Algumas expressões precisaram ser alteradas do formato EBNF, como a expressão de repetição {A} por A* ou a condicional [B] por B?
- As derivações foram identificadas com labels "`#label`" no final da regra para auxiliar na implementação futura da AST.
- Uma regra `param` foi criada para ser reutilizada na regra `params` e facilitar a implementação futura da AST.
- Nas derivações que utilizam um ID de tipo, um código extra foi adicionado para lançar uma exceção caso o ID não comece com letra maiúscula.

### 4.4. Identificação de Erros

Por padrão, o ANTLR identifica e se recupera de erros, processando a entrada até o fim. Além disso, existem alguns problemas no lançamento de erros relacionados ao Lexer e ao Parser especificamente.

Para resolver estas questões e poder identificar corretamente a classe `CustomErrorListener` foi implementada que redefine o método `syntaxError`, lançando a exceção "`ParseCancellationException`", que é identificada no código implementado do parser.

### 4.5. AST

O ANTLR não gera uma AST propriamente dita, e sim uma árvore de parse, que possui algumas características diferentes, como apontado na documentação da ferramenta. Assim, nós da AST foram definidos na pasta `ast/` e a classe `BuildAstVisitor` permite o caminhamento na árvore de parse gerada pelo ANTLR, acesso às derivações de cada regra da gramática e a criação manual dos nós que formariam a AST, retornando um SuperNode.

Como este não era o foco deste trabalho e os nós deverão ser alterados para a fase de interpretação, a estrutura não foi diretamente acoplada ao parser ainda. Para utilizar este código, basta **descomentar** as linhas 24-25 e **comentar** as linhas 27-29 do arquivo `FinalParser.java` e recompilar o projeto.

### 4.6. FinalParser

A classe `FinalParser` é utilizada na interface com os arquivos fornecidos previamente, implementando a classe `ParseAdaptor`. Nesta, cria-se um Lexer e um Parser fornecidos pelo ANTLR, o sistema de erros é configurado e ocorre a análise sintática do arquivo de entrada.

Caso a AST implementada esteja sendo utilizada, retorna-se um o nó “Prog” que representa a estrutura do programa lido. Caso contrário retornamos um nó padrão com a linha e coluna do token corrente do parser. Em ambos os casos, na ocorrência de erros, retorna-se `null`.

## 5. CONCLUSÃO

Com a finalização desta etapa do trabalho foi possível obter um maior entendimento do processo de análise sintática e ganhar experiência com uma ferramenta para este fim. O ANTLR é muito robusto e oferece diversas construções que auxiliam tanto na análise sintática, na análise léxica e em etapas posteriores, através da disponibilização de interfaces como Listeners e Visitors.