Universidade Federal de Juiz de Fora - UFJF - Departamento de Ciência da Computação

Teoria dos Compiladores - 2021/1

Alunos: Arthur de Freitas Dornelas 201735004
Vinicius da Cruz Soranço 201735003

1. Introdução

Nesta etapa do trabalho, foi desenvolvido um Analisador Léxico para a linguagem **lang**, dessa forma, foi utilizado a linguagem Java e o auxílio da ferramenta ANTLR 4. Essa ferramenta gera automaticamente os arquivos que são necessários para a implementação, facilitando as etapas que são mais repetitivas e simplesmente transforma o arquivo da linguagem em tokens disponíveis, gerando também algumas funções auxiliares para ajudar aos acessos dos tokens, além de ser muito mais eficiente para a evolução do compilador.

2. Analisador Léxico

Para definir o Analisador Léxico, foi preciso criar o arquivo **Lang.g4**, onde estão especificados quais são os tokens e seus tipos. A ordem das regras no arquivo tem importância, já que a primeira regra aplicável será a escolhida. Por exemplo, a palavra reservada “data” terá como tokenização “DATA”, porém ela poderia também ser considerada como “ID”, já que obedece a regra do mesmo, porém “DATA” tem prioridade nesse caso e aparece primeiro na lista de tokens no arquivo.

A partir disso, foi gerado pela ferramenta dois arquivos, o **LangLexer.java** e o **LangLexer.tokens.**

O **LangLexer.java** é responsável pelo Analisador Léxico em si. Essa classe é composta por alguns métodos, como **getRulesNames, makeRuleNames**, entre outras, que são basicamente utilizadas para o acesso e transformação dos tokens colocados no arquivo *.g4*. A outra parte importante do arquivo lexer é a estrutura de dados que é salvo os tokens. Ela é feita através de um autômato finito determinístico.

O **LangLexer.tokens** é responsável por salvar os tokens e ter um número associado a ele, de forma que diferencia os mesmo e facilite sua consulta através de inteiros e não string.

A outra classe que existe é a **LangCompiler.java**, é onde fica a *main* e basicamente faz a leitura do arquivo que contém o código que será analisado e executa o Lexer

em cima do mesmo. Ela também é responsável por printar na tela a sequência de tokens identificados.

3. Especificação dos Tokens

Data: 'data' ;
Print: 'print' ;
If : 'if' ;
Else : 'else' ;
Iterate : 'iterate' ;
True : 'true' ;
False : 'false' ;
New : 'new' ;
Read : 'read' ;
Return: 'return' ;
Plus: '+' ;
Minus: '-' ;
Mult: '*' ;
Div: '/' ;
Mod: '%' ;
And: '&&' ;
Or: '||' ;
Not: '!' ;
Less_than: '<' ;
More_than: '>' ;
Equal: '==' ;
Not_equal: '!=' ;
Open_parentesis: '(' ;
Close_parentesis: ')' ;
Open_bracket: '[' ;
Close_bracket: ']' ;
Open_curly_bracer: '{' ;
Close_curly_bracer: '}' ;
Accessor: '.' ;
Colon: ':' ;
Semicolon: ';' ;
Double_colon: '::' ;
Comma: ',' ;
Attribution: '=' ;
Identifier: Lowercase seguido por 0-N (Char, _ ou dígito).
TypeName: Uppercase seguido por 0-N Lowercases.
Integer: 0-N dígitos.
Float: 0-N dígitos seguidos por um ponto seguido por 1-N dígitos.
Literal: um Char entre ‘’ ou os valores ‘\n’, ’\t’, ‘\b’, ‘\r’
Char: Uppercase ou Lowercase.
Boolean: valores verdadeiro ou falso. (true ou false)
Null: é o valor nulo. (null)

WhiteSpace: término de linha ou ‘\t\f’.
LineTerminator: término de linha, reconhecido como ‘\r’, ‘\n’ ou ‘\r\n.
Comentário:

a. ao encontrar um ‘{-’ inicia a leitura do comentário de bloco e até encontrar um ‘-}’ ignora todos os valores lidos.
b. ao encontrar um ‘--’ inicia o comentário da linha e até encontrar um **Line Terminator** ignora todos os valores lidos.

4. Compilação

Para executar o analisador léxico basta entrar na raiz do projeto e rodar o *build.sh* pela linha de comando ou executar os seguintes comandos em sequência:

- *java -jar ./lang/lexer/grammar/antlr-4.8-complete.jar -o lang/lexer/grammar/antlr -package lang.lexer.grammar.antlr -visitor ./lang/lexer/grammar/Lang.g4 -Xexact-output-dir*

- *javac -cp .:lang/lexer/grammar/antlr-4.8-complete.jar lang/LangCompiler.java*

- *java -cp .:lang/lexer/grammar/antlr-4.8-complete.jar lang/LangCompiler -bs*

Com isso, o analisador léxico vai ser gerado, o nosso código vai ser compilado e executado, imprimindo os tokens na tela. Ao executar o build ou os comandos, pode ser que apareça o seguinte erro:

error(99): ./lang/lexer/grammar/Lang.g4::: grammar Lang has no rules

O mesmo deve ser desconsiderado, pois se trata da ausência de dados no arquivo *Lang.g4* que estão relacionados a geração do analisador sintático, o que não nos interessa no momento.

5. Testes

O arquivo com o código para ser feita a análise léxica será o **“test.lang”**, que está na raiz do projeto. Para realizar um teste com outra entrada deve-se modificá-lo e mantê-lo no mesmo local e reexecutar o *build.sh* para o funcionamento correto.