Implementação das etapas de compilação

Marcelo Elias Knob[1], Marcio Junior Rios[2]

[1 2] Universidade Federal da Fronteira Sul

[1 2] Curso de Ciência da Computação – Chapecó, SC – Brasil

[1]marceloknob013@gmail.com, [2]marciojrrock@hotmail.com

Abstract. *This meta-article aims to describe how the construction the project of the compilation steps, in which the minimum expected result of the work was implemented, that is, the lexical and syntactic analyzer. Python3 is used as the programming language for the creation of the algorithm, and code fragments and test outputs are shown in order to better elucidate the subject. It also contains brief explanations about each build step, in order to make the reader aware of the subject.*

Key-words: *compilation, step, analyzer, lexical, syntactic, Python3*

Resumo. *Este artigo tem por objetivo descrever como foi a construção do projeto das etapas de compilação, no qual foi implementado o resultado mínimo esperado do trabalho, ou seja, o analisador léxico e sintático. Utiliza-se python3 como linguagem de programação para a criação do algoritmo e são apresentados fragmentos de códigos e resultados das saídas dos testes a fim de elucidar melhor acerca do assunto. Contém também breves explicações sobre cada etapa de compilação, com o intuito de deixar o leitor inteirado no assunto.*

Palavras-chaves: *compilação, etapa, analisador, léxico, sintático, Python3*

1. Introdução

Por meio do entendimento de determinados conteúdos do componente curricular Construção de compiladores do curso de Ciência da Computação tem-se o gerenciamento da avaliação do processo de ensino e aprendizagem. Nele consiste uma avaliação a qual se tem a elaboração de um projeto que tem como descrição a construção das etapas de compilação para uma linguagem de programação hipotética. Essas etapas são divididas em Análise Léxica, Análise Sintática, Análise Semântica, Geração de código intermediário e Otimização, mas as que foram implementadas a seguir são as duas iniciais. Como é necessário de um AFD (Autômato Finito Determinístico) para a implementação dessas etapas foi utilizado do trabalho anterior para a execução deste com algumas correções.

2. Referencial teórico básico

Para a elaboração da aplicação é necessário o conhecimento de determinados conteúdos, os quais temos os conceitos de AFD, analisador léxico e sintático. Com isso, temos primeiramente o entendimento de um AFD que é uma máquina de estados finita que aceita ou que rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada. A parte do “Determinístico” tem a ver com a unidade do processamento. Depois começa a etapa inicial da construção de compiladores, o qual se tem o analisador léxico que é responsável pelo processo de analisar a entrada de linhas de caracteres (como o código-fonte) e produzir uma sequência de símbolos (tokens), que podem ser manipulados mais facilmente por um parser. Já para a segunda etapa temos o analisador sintático que é responsável pelo processo de analisar uma sequência de entrada para determinar sua estrutura gramatical segundo uma determinada gramática formal.

3. Especificação e implementação da solução

Já dito anteriormente, o algoritmo foi criado utilizando o python3 como linguagem de programação. Os seguintes subtítulos apresentarão a implementação da solução do algoritmo e seu funcionamento.

3.1 Leitura dos Tokens e da Gramática Regular

Foram utilizadas listas para guardar os símbolos da linguagem que foram retirados dos tokens e das gramáticas. É utilizado uma lista de dicionários para armazenar as produções.

```
1   if
2   else
3   while
4   equal
5   obracers
6   cbracers
7   sub
8   sum
9   mult
10  div
11  <S> ::= a<A> | e<A> | i<A> | o<A> | u<A> | 0<B> | 1<B> | 2<B> | 3<B> | 4<B> |
    5<B> | 6<B> | 7<B> | 8<B> | 9<B>
12  <A> ::= a<A> | e<A> | i<A> | o<A> | u<A> | &
13  <B> ::= 0<B> | 1<B> | 2<B> | 3<B> | 4<B> | 5<B> | 6<B> | 7<B> | 8<B> | 9<B> | &
```

3.2. Criação do Autômato

É utilizado uma lista de estados que tem a função de armazenar todos os estados criados no AF. É utilizado uma lista de dicionários para armazenar as produções da tabela, que estão relacionadas aos estados pelo índice na lista.

3.3. Determinização

Foram utilizadas duas estruturas, uma lista para armazenar os não terminais que foram transformados em estados e um dicionário para armazenar a relação de {(não-terminal): (estado no autômato)}

3.4 Analisador Léxico

Cada linha do arquivo é salva em uma lista, em seguida, é separada cada string da linha e começa o reconhecimento de cada string. O reconhecimento é dado lendo cada caractere da string e realizando movimentos no AFD até chegar no estado final onde é reconhecido a string. Todos os reconhecimentos são salvos na Tabela de Símbolos que armazena a linha no código-fonte, o estado final e o rótulo.

3.5 Gramática Livre de Contexto (GLC) e Gold Parser

A GLC criada que foi utilizada no Gold Parser para a geração da tabela LALR está dada na imagem abaixo.

```
"Case Sensitive" = False
"Start Symbol" = <S>
<S> ::= <var> <ops>

<var> ::= '*46' 'equal' '*46' <var> | '*46' 'equal' '*47' <var>| <>

<oprel> ::= 'sum''*47' | 'sub''*47' | 'div''*47' | 'mult''*47' | 'sum' '*46' |
'sub' '*46' | 'div' '*46' | 'mult' '*46' | 'equal'<igual> | <>

<ops> ::= <if> | <while> | <>

<if> ::= 'if'  '*46'<oprel> 'obracers' <var><ops> 'cbracers' <else>

<else> ::= 'else' 'obracers' <var><ops> 'cbracers'

<while> ::= 'while' '*46'<oprel> 'obracers' <var><ops> 'cbracers'

<igual> ::= 'equal' '*46' | '*46' | 'equal' '*47' | '*47'
```

3.6 Analisador Sintático

Foi criada a pilha com o estado inicial ‘0’ e foi extraída a fita da TS. Então é utilizada a tabela LALR em conjunto com a pilha e a fita para realizar as ações necessárias (empilhar, reduzir, salto e aceita).

4. Conclusão

É possível concluir que o maior desafio desse trabalho foi entender o que deveria ser feito, como fazer para que o algoritmo fosse o mais genérico e correto possível, da mesma forma como ocorreu na implementação do projeto de construção da AFD. Como o trabalho anterior foi implementado com a linguagem de programação python3 o mesmo continuou nesse, tanto para focar mais tempo na construção do compilador e não aprendendo uma linguagem nova, como pelas facilidades que a linguagem oferece para manipulação de arquivos, algo muito utilizado nesse trabalho. A partir da GLC temos o código-fonte e assim os testes, os quais provaram um ótimo funcionamento das etapas implementadas e demonstrando um bom prognóstico. Por fim é importante salientar a enorme contribuição do trabalho no entendimento do assunto, o qual ajudou muito a entender como e o que ocorre nas etapas de compilação, facilitando o entendimento de como o código-fonte é analisado e as informações que lhes são utilizadas.

Referencial bibliográfico

Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2001). Introduction to Automata Theory, Languages, and Computation 2 ed. [S.l.]: Addison Wesley. ISBN 0-201-44124-1. Consultado em 2 de dezembro de 2019.